SAMS

26167

VIC 20

Games, Graphics, and Applications

David D. Busch

VIC 20:
Games, Graphics, & Applications

Ⓟ1983 David D. Busch

Cat. No.
26513

Side 2
(Enhanced
Versions of
Programs)

Howard W. Sams & Co., Inc.
4300 West 62nd Street, Indianapolis, Indiana 46268 U.S.A.

SAMS 22189

VIC 20™

Games, Graphics, and Applications

David D. Busch

VIC 20 GAMES, GRAPHICS, AND APPLICATIONS

David Busch is probably best known for his Kitchen Table, Inc. computer humor series, which has appeared in magazines like 80 Micro and Softside. In a more serious vein, he has been a reporter, observer, and participant in the computer industry for nine years. He has written more than 300 computer-oriented articles for most of the major trade and personal computer magazines, and is a contributing editor to Interface Age magazine.

His VIC 20 credentials include a stint as reviewer of VIC 20 hardware and software for Creative Computing magazine, and several full-length discussions of VIC 20 features in other leading personal computing publications. He is already at work on another book of VIC 20 games, graphics, and applications.

Besides owning two VIC 20 computers (with "every hardware accessory made for it"), Busch has three Radio Shack computers, including a TRS-80® Model 100; a Sinclair ZX80; and a Xerox 860.

Busch's other interests include music (Hank Williams, Jr., the Beatles, Beethoven, and The Clash), Charlie Chaplin lore, and collecting travel guides to Spain published before 1900.

VIC 20 GAMES, GRAPHICS, AND APPLICATIONS

By David D. Busch

Howard W. Sams & Co., Inc.
4300 WEST 62ND ST. INDIANAPOLIS, INDIANA 46268 USA

Copyright © 1983 by David D. Busch

FIRST EDITION
SECOND PRINTING—1984

All rights reserved, No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

International Standard Book Number: 0-672-22189-6
Library of Congress Catalog Card Number: 83-50374

Edited by *Welborn Associates*

Printed in the United States of America.

Preface

The Commodore Business Machines VIC 20 has become one of the most — if not THE most — popular microcomputers in the United States and overseas. The reasons are obvious. The VIC 20 is a real, full-functioned microcomputer. It has a professional-style keyboard instead of the calculator-type or flat membrane keyboards common on under-$100 computers. Lower-case letters and a full graphics character set are built in. Its BASIC is a full-featured version of Microsoft's BASIC. Students can learn programming on the VIC 20, and move up to other Commodore computers, or transfer their knowledge to many Radio Shack, Apple, or similar computers. Yet, with a joystick/paddle/light pen port and full-color capabilities, the VIC 20 is a games machine second to none.

Those who want to learn BASIC programming will find that the special features of the VIC 20 are easy to use. The machine has many powerful techniques available, such as user-definable character sets, four musical voices, a real time clock, and a multitude of color and graphics capabilities. Some of these capabilities can be employed through add-on programs or accessories, such as Commodore's Gamegraphics Character Set editor, and the Super Expander Cartridge.

Those who want to learn more about BASIC programming will prefer to tackle these fascinating aspects of the VIC 20 on their own, however. Any of them can be accessed from BASIC using simple commands.

The best way to learn exactly how to use these features is through typing in, examining, modifying, and using the 20 simple BASIC programs in this book. Most can be run in the 5K unexpanded VIC. All have been tested and are ready to RUN. Simple games, through full-blown, arcade-style joystick extravaganzas fully demonstrate the capabilities of this full-grown computer that is priced like a video game. VIC GAMES, GRAPHICS, AND APPLICATIONS will show you some of the intermediate and

advanced tricks of BASIC programming, as they relate to the VIC 20's special features. You'll find out how to make simple sounds, use joysticks to move objects around on the screen, and time functions with the real time clock. Best of all, the learning process will be fun, because you will have 20 working games and applications programs to key in, examine, and play with when you are finished.

David D. Busch

Trademarks

Apple is a registered trademark of Apple Computer, Inc.
BASIC is a registered trademark of the trustees of Dartmouth College.
Radio Shack is a registered trademark of Tandy Corporation.
VIC 20 is a trademark of Commodore Electronics Limited.

Synopsis of Programs

1. **Computer ESP** — Watch the VIC 20 answer questions from party-goers. Is the magician signaling the computer somehow? Changing the screen/border combinations introduces you to the mystery of POKEs to memory. Learn the Black Magic of MID$, and how to use it to remove selected characters from a longer word or phrase.

2. **Mind Reading** — Can the operator of the VIC 20 guess which object has been selected by the audience — WITHOUT touching the computer? Teaches problem-solving techniques through examining situations in unusual lights. What seems to be may not always be the case! Also, how does the VIC 20 arrive at random numbers — and are they truly random?

3. **Autocost** — How much does your car REALLY cost to operate? This VIC 20 program will let you know. Shows how to use long variable names to make program flow clearer, debugging easier. Learn a programming trick to format your dollars-and-cents output neatly.

4. **Space Command** — Subroutines, and how they can make a program easier to read, are explored in this game. The goal is to return the rebel fleet to base before running out of photon torpedoes or fuel. But watch out! The Quillons may attack on the way.

5. **Album Timer** — Hmmm . . . What length tape to use? Simply key in minutes and seconds for each selection. This program totals the time. Learn how to embed graphics characters in PRINT statements to dress up even the simplest program.

6. **Cost Estimator** — Shows how to use VIC 20 to calculate roughly the materials needed for many home framing projects. Use of descriptive variable names is demonstrated. Introduces you to the computer assisted drafting concept. Engineers use

sophisticated versions of programs like this to compile bills of material for construction projects.

7. **Reaction Timer** — Uses for the VIC 20 built in real time clock. Name of special function key flashes on the screen suddenly, and each player must hit a key as quickly as possible. Reaction time, in 1/60 of a second, best times, and winner are displayed. In this chapter, learn how to "read" the special function keys and make them do your bidding.

8. **Stock Market** — You can sell short, buy low, sell high, and react to market trends with this fast moving game. Spend more money than you have — but catch up before you quit. You probably won't learn much about Wall Street from this game, because, unlike life, it is possible to go away with the same amount of money with which you began.

9. **Home Bulletin Board** — Leave a message for a family member. Protect with password, or allow anyone to read. VIC 20 beeps when messages await. Further use of the real time clock, showing how to set the current time for later reference. Also shows the application of string arrays for message storage.

10. **Kitchen Timer** — Lets your VIC 20 time various recipe steps easily and automatically. Once you've set the correct time, it's necessary to have the computer check it against a future limit. This program shows you how.

11. **Bingo** — Automatic Bingo for the VIC 20 demonstrates utility of random number generation and POKING graphics to screen. More discussion of the VIC 20's video memory map.

12. **Floor Planner** — Computer assisted design, VIC style. VIC 20 draws floor plan on the screen. Move your furniture the easy way and see how your office/home will look. This introduction to reading the joysticks in a BASIC program teaches how to use them in a nongame program.

13. **Cookie Shop** — A simple game that teaches basic principles of free enterprise to children. Program provides extensive use of VIC 20 graphics, using only symbols embedded in BASIC program statements.

14. **Black Book** — The data statement data base. Add information without resorting to either disk or cassette files. The method

is faster than using cassette files and as easy as disk. Simply SAVE your entire BASIC program with the DATA entered in new program lines.

15. **Motor Race** — Define your own character sets with the VIC 20. Shows how to replace "unneeded" alphanumerics with user-designed racing car and pylons. Also demonstrates use of joystick.

16. **VIC Organ** — Using the VIC 20 as a music synthesizer. Tunes can be entered into memory. Voices are changed by pressing special function key. Play them back by pressing another. Good graphics in this exercise portray the keyboard on the screen. Thorough grounding in VIC 20 music capabilities.

17. **Barrier Run** — A joystick game that is like some you might have seen, but with a twist. Attempt to move your growing "snake" past increasing maze of barriers. Score is calculated from elapsed time. To make things interesting, computer randomly demolishes holes through various walls that you can escape through.

18. **POP !** — Arcade-style joystick game with moving targets, arrow-firing "base" and POKE graphics reveals intricacies of VIC 20 games possible using only BASIC commands. Uses all of the features discussed so far in this book.

19. **Auto Writer** — Another useless program, unless you let it run for 100 years or so! Generates Pulitzer Prize winning novels by choosing letters at random. The only problem is separating the wheat from the chaff.

20. **Urban Renewer** — Wondering what to name your subdevelopment? This program, just for fun, will give you 20 suggested names in 20 seconds. A silly program, but fun to watch.

Contents

Introduction

Chapter	*Program*	*Page*
1.	Computer ESP	17
2.	Mind Reading	23
3.	Auto Cost	29
4.	Space Command	35
5.	Album Timer	45
6.	Cost Estimator	51
7.	Reaction Timer	57
8.	Stock Market	63
9.	Bulletin Board	69
10.	Kitchen Timer	75
11.	Bingo	81
12.	Floor Planner	85
13.	Cookie Shop	91
14.	Black Book	97
15.	Motor Race	101
16.	VIC Organ	109
17.	Barrier Run	113
18.	POP !	117
19.	Auto Writer	123
20.	Urban Renewer	127

Introduction

THE PROGRAM LISTINGS

The VIC 20 uses a variety of special graphics symbols within its programs to indicate various screen printing codes. Not all of these are chosen as simple logical representations of the codes they symbolize. For example, a reverse R does mean CTRL (control) R, which turns on the reverse character mode. However, "reverse off" is shown on the screen by a reversed underline character. Some other codes are even more cryptic. The clear screen symbol is a heart, while the reverse Q indicates the cursor down key. Color codes are entered at the keyboard by pressing CTRL and a number key with the appropriate color printed on it. These are shown on the screen as, for example, a reverse English pound sign for CTRL-3 (RED).

While these symbols are clear enough on the screen, they do not reproduce especially well, even when printed by a high-density graphics printer. For this book, I have more or less adopted the conventions used by several magazines, and substituted abbreviations for the special symbols in the program listings. Instead of a heart symbol, "(CLR)" is used. The cursor down key is not represented by a reversed Q; "(DWN)" is employed instead. Where a key is repeated several times, the number of repetitions is shown: "(02 DWN)" or "(12 CBM – +)." When the control (CTRL), shift (SHF), or Commodore (CBM) keys are indicated, they are followed by a dash, and then the actual key to be pressed. "(CBM–+)" would be activated by pressing the Commodore key and the plus key simultaneously. "(CTRL-Q)" would be control-Q, while "(SHF-P)" would be shift P. Another convention for program listings — packing as many characters on a line as possible — has been ignored. There are several reasons why many of the listings for VIC 20 programs that you see in magazines or books are compressed so tightly. The 5K VIC 20 has

only about 3.5K of memory available for user programs. Therefore, it is frequently necessary to use as little space as possible in order to fit all the program within those constraints. In addition, a packed program can be a little easier to type, because the typist does not have to watch to see where spaces go. He or she can simply type a stream of characters nonstop. The VIC 20 interpreter ignores the spaces between keywords, or between keywords and numbers or operators anyway.

However, such compressed programs are a great deal more difficult to read when debugging, or simply examining the code to learn how the program works. Because of the tutorial nature of this book, I have chosen to "open up" the listings. Spaces are used liberally between words. In many cases, only one statement is used per line. More than one statement are generally used only when the statements are very simple and go together, such as:

```
10 GET A$: IF A$="" GOTO 10
```

At other times, statements are placed on the same line because they follow a true IF . . . THEN statement. As an example:

```
10 IF A=1 THEN PRINT "A=1":GOTO 30
20 PRINT "A DOES NOT EQUAL 1"
30 END
```

All substantive program lines are numbered increments of 10, and thus end in a zero. Remarks, other than the title block at the beginning of the program, are numbered in increments of five; and you can safely delete them when typing in the program. It's probably a good idea to retain the title block, if only to make it clear which program you have. It's entirely possible to give a program the wrong name when saving it to tape or disk. But, barring catastrophies, the title block should be accurate.

If you wish, you can also delete spaces that are NOT contained within prompts (i.e., IFA=2THENPRINT"DON'T DELETE THESE SPACES!"). Be especially careful to enter all the spaces contained within the prompts, as they affect screen formatting. For example:

```
10 PRINT"(RVS)(BLK) (RED) (GRN) (OFF)"
```

will print a black, red, and green square on the screen — but leaving the spaces out will print nothing!

Some of the programs in this book **DO** take up more than 5K of space. It seemed a shame to neglect those of you who have purchased one of the inexpensive memory expansion cartridges for the VIC 20. We kept the latter group in mind further by writing nearly all of the programs so that they will function regardless of memory configuration. There will be no need to pull your memory expansion cartridge in order to run some of the games in this book. And that's more than can be said for many commercial programs on the market.

SCREEN SYMBOL EQUIVALENTS

(CTRL)	Control	Hold down control key and indicated key simultaneously.
(SHF–)	Shift	Hold down shift key and indicated key simultaneously.
(CBM–)	Commodore key	Hold down Commodore key and indicated key simultaneously.
(BLK)	CTRL-1	Black
(WHI)	CTRL-2	White
(RED)	CTRL-3	Red
(CYN)	CTRL-4	Cyan
(PUR)	CTRL-5	Purple
(GRN)	CTRL-6	Green
(BLU)	CTRL-7	Blue
(YEL)	CTRL-8	Yellow
(REV)	CTRL-9	Reverse On
(OFF)	CTRL-0	Reverse Off
(CLR)	SHF-CLR	CLR
(DWN)	CRSR	Cursor Down

HOW TO USE THIS BOOK

Unless you are an advanced programmer, it is recommended that you start at the beginning and work forward. Admittedly, some of the more interesting programs are in the middle or final chapters of the book. However, each of the various special features of the VIC 20 explained is introduced one at a time, in a specific order. The gradual progressive approach makes using complex capabilities quite a bit easier. For example, "Reaction" introduces the concept of the VIC 20's built-in clock by showing how to measure a given time span in $\frac{1}{60}$ second intervals. "Bulle-

tin Board'' expands on the idea and demonstrates how to set the clock to the current time. ''Kitchen Timer'' then finishes with a module that allows you to set an ''alarm'' for some future time and use the real time clock to compare that target time with the current time.

Joysticks, graphics, and the use of sound are also revealed gradually and logically. While this book does attempt to provide an introduction to many VIC 20 features that look complex, it is not a first programming book. It assumes you know something about FOR . . . NEXT loops, use of GOTO, and other simple statements. Ideally, you should have written several programs previously on your own, and now be ready to take on more advanced programming and tips. However, this book does explain how things work, especially as they relate to the differences between the VIC 20 and other computers.

If you already can program some other personal computer, this is your shortcut to VIC BASIC proficiency. If this is your first taste of real games, graphics, and applications programming, look forward to a concise, entertaining introduction.

CHAPTER 1

COMPUTER ESP

Type: Party Game, any number of players
Size: 800 bytes, for any size VIC 20

When the first home computers became available, much of what they could do seemed like something close to magic. Adding up the totals in a checkbook could be done in seconds! Letters could be written error-free, and printed out neatly as many times as desired! Computers, amazingly enough, could even do income taxes! Although they themselves are never required to pay them.

If you have had your VIC 20 for some time, its mysteries are less unfathomable than before. This book is aimed at revealing some of the secrets of the VIC, in an attempt to add it to the ranks of computer wizards. We'll start off, then, with some computer magic that can be used to liven up a party.

Even those who understand the workings of your VIC 20 will be mystified by this demonstration of Computer ESP. The program also provides us with the opportunity to explore one of the simplest special "tricks" you can do.

When the VIC 20 is turned on (*powered up* in computer slang), it will display blue letters on a white background, with a cyan border. You learned quite early in your experience with the VIC that you can change the color of the letters, numbers, and graphics as you type, simply by hitting the CTRL key and one of the number keys on the top row. CTRL-1, for example, will change any new screen symbols that appear to black instead of

blue. CTRL-2 will switch to white, which makes the symbols, of course, invisible on the white screen. You can also change the screen background color, as well as the border color, but not quite so simply as just striking a key. To do this, it is necessary to "POKE" a number into a specific memory location. POKEing is one of the most powerful tools available to you from BASIC and will be used throughout this book.

If you had 65,000 file cabinets at work, you might ask an associate to go peek in file number 35,232 and tell you what was in there. What you did next might hinge on that information. For example, if you commonly hid your lunch in file cabinet 35,232 but found, on peeking, that it was empty, your course of action might be to make reservations at a local restaurant.

Assume, for a moment, that it is not your responsibility to hide the lunch in the cabinet. Say that your spouse liked to surprise you by sneaking into your office and poking a lunch treat into the cabinet for you. Even though the computer does not eat lunch, it allows the user to access many of its own memory's filing cabinet "drawers" in a similar manner.

The VIC 20 has available to it 65,535 separate memory locations. Some of these "drawers" are occupied by read only memory (ROM), which contains fixed data, such as the BASIC language itself, that is not destroyed when the computer power is turned off. These might be thought of as drawers with glass covers. The user can see what is in them, but cannot change the contents. Other memory locations are occupied with random access memory (RAM), which can be changed by the user. The most common application for RAM is to fill it with the BASIC program that you write.

However, some RAM locations store information that is used by the computer operating system. These locations are loaded by the computer with data when the computer is powered up; but, since the memory is "writable" (unlike ROM, which can be "read" by the computer, but not changed), the user may alter the values to suit special needs.

One such location is at address 36879. This might be thought of as the 36879th "box" in the VIC 20's memory. By typing PEEK and the memory location, you can look at the contents of any storage compartment in the VIC 20's memory. It is also possible to POKE numbers into random access memory.

If you will type PRINT PEEK(36879) on power-up, you will see that the computer has loaded the value 27 there. Appendix E of

the "Personal Computing" guide furnished with the VIC 20 has a chart that shows what numbers stored in 36879 will reproduce what screen/border color combinations. Scan the chart to 27 and confirm that this produces a white screen with a cyan border. Other color combinations are possible just by POKEing some other number into that memory location. With a dark screen color, you can even use white letters.

COMPUTER ESP uses changing screen/border color combinations as a distracting feature to draw the audience's attention away from what is really happening. Briefly, the trick allows the computer to answer simple questions from the bystanders, with the clever aid of the computer operator. To run the program, boast that the computer is able to communicate through various means, by telephone, hard-wiring, or other electromagnetic media (the more scientific the explanation, the more the audience will be distracted). Explain that under certain conditions the computer can also receive telepathic signals. To demonstrate, the operator will sit down and ask the computer to answer a question from the onlookers. Sure enough, an answer appears right on cue.

How is the trick done? While commands such as "Try to guess this one, next" *APPEAR* to be typed by the operator, the magician is actually typing the answers at that time. What appears on the screen are actually messages that have been embedded in the program, and which are revealed a character at a time as the operator types.

To the properly distracted audience, it appears as if the screen is echoing the typing. We are so accustomed to seeing words appear on paper or screen that few of us actually watch someone's fingers as they type.

To gain maximum effect from this trick, it would help if the magician were a touch typist who could turn to make eye contact with the audience, direct their attention at the screen, or do other things that will keep them looking anywhere but at his/her hands.

To help in the distraction, the program changes screen/border color combinations at several opportunities. I chose to have the screen remain light yellow, but with the border changing each time. So, a basic value of 247 (one less than the first number in the "light yellow" row on the combinations chart) is assigned to a variable named "SCREEN," in line 60. Then, a FOR . . . NEXT loop from 1 to 8 commences. This loop will repeat eight times to

allow eight messages to be answered on the screen. Each time the loop is incremented (stepped off another number), line 80 POKEs the value of SCREEN+N into the magical 36879 location. On the first time through the loop, N will equal 1, so 248 will be POKEd into the box, producing a light yellow screen with a black border. On the second time, when N equals 2, 249 will be POKEd, producing a white border.

That takes care of the screen. What about the messages? Line 120 will GET a single character from the keyboard. GET is something like INPUT, except that INPUT will wait for you to press RETURN. You may then input a whole string of characters. GET, on the other hand, will accept only one character, and only at the moment the computer runs that particular program line. If no key is pressed, the computer simply goes on to the next statement on that line, or the next line.

Accordingly, GET statements are usually put in some sort of loop, which repeats until a character is in fact entered. On any trip through the loop, if no key is pressed, then the variable assigned to GET, in this case KBD$, will equal nothing or "". The two quote marks with no space in between are used to signify a "null string," with a "string" being any series of characters.

If KBD$ does in fact equal nothing, then the program, in line 130, directs control back to 120 to try again. This process repeats thousands of times a second, until the operator *DOES* happen to press a key. In this case, the program drops down to line 140, where it checks to see if RETURN, which the computer sees as CHR$(13), has been pressed. The VIC 20 can produce 255 alphanumeric and graphics characters, each of which is given a number in the range 1–255. The capital letter "A," for example, happens to be the 65th, and can also be called CHR$(65). (Type PRINT CHR$(65) to see what happens.) The carriage return character has been assigned number 13, and that's what the computer looks for in line 140. If it finds a carriage return, the program goes to line 200.

If not, it next checks, in line 160, to see if the key pressed was a period. In operating the program, the magician touch-types the answer, ending it with a period. He or she can continue typing, however, to complete the message on the screen. After all, the answer being entered and the screen message will rarely be exactly the same length.

Whenever a period IS entered, though, a flag is set (line 150), which tells the computer to ignore all additional input. The operator can simply type nonsense characters.

Until that point, any character typed in will be added onto those which have been entered previously. As each key is pressed, the value, KBD$, is added onto the answer, AN$, in line 160, but only if FLAG does NOT equal 1.

Where does the computer get these messages? Lines 300 through 370 have the messages stored as DATA lines. Each message in turn is assigned to MESSAGE$ in line 110 by a READ statement. One character from the message is displayed at a time, extracted by the MID$ function. MID$ is a string-handling tool that allows you to take one (or more characters) from a larger string. It uses the form MID$(LARGER STRING, POSITION TO START, NUMBER OF CHARACTERS TO TAKE). A counter, CU, keeps track of how many characters have been entered. It then is used to tell the program to take a character beginning at CU in MESSAGE$. The first time line 180 is called, the first character is printed. The second time, CU will equal 2, so the second will be printed, and so forth.

When RETURN is pressed, the computer branches to line 200, where it prints the answer, AN$, then asks that any key be pressed to go on to the next question.

Another GET KBD$ loop repeats until a key is pressed. At this point, the variables used, such as AN$, CU, and FLAG, are returned to null or zero, and NEXT N sends the loop back to line 70 for another trip through.

All magic tricks are best when the audience is begging for more, and so this one should not be repeated too often. Eventually someone will catch on and watch the magician's fingers. Or, someone will suggest a message to be typed in and the operator will be unable to comply. The eight messages provided are probably the maximum number of times it should be done for any audience. You can substitute messages of your own, by the way.

In case the magician does get carried away, there is no danger of being embarrassed by an OUT OF DATA message. After all eight DATA lines have been read (the FOR . . . NEXT loop has gone through all eight trips), a RESTORE in line 280 will re-initialize the program, and send everything back to the beginning.

```
10 REM ****************
20 REM *               *
30 REM * COMPUTER ESP *
40 REM *               *
50 REM ****************
60 SCREEN=247
70 FOR N=1 TO 8
80 POKE 36879,SCREEN+N
90 PRINT"(CLR)(02 DWN)"
100 PRINT TAB(5)"(REV)(RED)M(CYN)A(PUR)G(GRN)I
                (BLU)C(OFF)(REV)(BLK)S(RED)E
                (CYN)E(PUR)R(OFF)(BLU)(02 DWN)"
110 READ MESSAGE$
120 GET KBD$
130 IF KBD$="" GOTO 120
140 IF KBD$=CHR$(13) GOTO 200
150 IF KBD$="." THEN FLAG=1
160 IF FLAG<>1 THEN AN$=AN$+KBD$
170 CU=CU+1
180 PRINT MID$(MESSAGE$,CU,1);
190 GOTO 120
200 PRINT:PRINT"(DWN)";AN$
210 PRINT"(03 DWN) (REV)PRESS ENTER FOR "
220 PRINT" (REV)ANOTHER QUESTION"
230 GET KBD$:IF KBD$="" GOTO 230
240 AN$=""
250 CU=0
260 FLAG=0
270 NEXT N
280 RESTORE
290 GOTO 10
300 DATA " GREETINGS, COMPUTER  SAGE."
310 DATA "TRY TO GUESS THIS ONE,NEXT."
320 DATA "NOT SO BAD FOR A COMPUTER."
330 DATA "WOULD YOU LIKE ANOTHER?"
340 DATA " TELL US THE ANSWER,  PLEASE."
350 DATA "OKAY, DO IT AGAIN."
360 DATA "ANOTHER PERFECT SCORE."
370 DATA "LAST QUESTION. READY?"
```

PROGRAM LISTING 1. COMPUTER ESP

CHAPTER 2

MIND READING

Type: Party Game, any number of players
Size: 1300 bytes, for any size VIC 20

Like the taste of magic? Need a trick to follow up after the demonstration of Computer ESP? Well, since it seems so easy to have the computer read your audience's mind, how about turning the tables and having the magician read the computer's "mind"?

MIND READING will allow the audience to type in the names of eight objects in the room, while the magician is out of sight and hearing. Then, one of the objects may be selected by one person — by vote of the audience — or even by the computer. In fact, the computer can select one object secretly so that even the audience does not know.

Then, when the magician returns, he or she will tell which object was selected WITHOUT touching the computer. If no one in the audience knows, because the computer has selected the object, then, truly, the magician must have read the mind of the machine. The object chosen can be revealed without the mystic sage ever touching the keyboard.

Magic again? Not if you know the secret. In this trick, we use the screen/border combination as both a distracting feature and as the tipoff to the magician! The colors change from time to time. However, when the screen clears and the sage is asked to tell which object was chosen, it is the color of the screen border at that time which reveals the object.

If you scan the VIC keyboard, you will notice that each of the number keys has a color associated with it, 1 for black, 2 for white, and so forth. The color of the VIC 20 border will be the SAME as the number of the object chosen. If the border is black, the selection is object no. 1, a white border will indicate that the audience or computer has chosen object no. 2, and so forth.

On many color tv's and some monitors, the colors are difficult to differentiate. Red may look a lot like purple, and green or cyan can be tricky to tell apart. Usually, the television can be adjusted to provide purer colors and better contrast. But, with a magic trick like this, a casual error could be disastrous. To make things easy, the program prints color bars on the screen in the same order as they appear on the keyboard. Since the border is next to this color bar, it is quite simple to compare the border color with the color bar, and determine precisely whether the border is blue or cyan, for example.

This program also introduces the concept of RND, or the choice of a random number by the computer. Most of the program is quite simple. A FOR . . . NEXT loop of from 1 to 8 allows the bystanders to enter the names of eight objects, which are stored in a string array, OBJECT$(n). A string array with one dimension (such as this one) might be thought of as a long row of boxes, each capable of storing a string of characters. If this row is 10 or less boxes long, the computer will handle setting it up automatically. If the row is 11 or more long, we need to define the array, or "dimension" it with a DIM statement.

DIM is used to tell the program how long the array will be, and is used in the form DIM NAME OF ARRAY(SIZE). We might enter DIM OBJECT$(20) to indicate that we want a string array with a single row with 20 "boxes." Arrays with more than one dimension ("columns" as well as "rows") are also used in some programs.

Since the array OBJECT$(n) is smaller than 10, there is no need to DIMension it. When memory space is tight, it is sometimes a good idea to DIMension arrays even if they include fewer than 10 elements. Otherwise, the computer reserves memory space for the boxes that will not be used. For example, the following program line would save a significant number of bytes when used in an unexpanded VIC 20:

```
10 DIM A$(2,2),B$(4),C(3,3),D(7)
```

In the interest of memory economy, you also should know that the first "box," or element of an array is always 0 (i.e., OBJECT$(0)), while the second is OBJECT$(1), the third OBJECT$(2), etc. It does no harm to leave a given box empty, however, and for the sake of clarity, many programmers simply write software as if 1 were the first box.

Once all eight objects have been named, the program pauses while a GET A$ loop repeats until a keyboard character is entered (line 320). Then the value of the character entered is determined (line 330), and this variable, CH, examined to make sure that it is within the allowable range 1–2. If it is smaller than one, or larger than two, control goes back to line 320 to await another character.

This sort of routine is useful to keep neophytes from entering an incorrect value. It won't accept any character other than the numbers 1 or 2. By changing the limits, you can alter the program to accept only menu choices from 1 to 8, and so forth, within the desired range.

If the choice made is number 2 (CH=2), then the computer chooses a random number between 1 and 8 in line 350. The VIC 20, when it encounters the statement "RND(1)" will choose a number larger than zero, but smaller than one. This might be .562391 or .29171, or some other decimal fraction. However, we want whole numbers in the range 1–8. To produce these, we multiply by the largest number we want to reach, and add one. For example, RND(1)*8 will produce real numbers larger than zero (but less than one) to numbers larger than seven, but less than eight. Adding one to any of these will give us random numbers between 1-plus and 8-plus. Taking the integer portion of the number arrives at whole numbers in the desired 1–8 range.

Are the numbers chosen truly random? Strictly speaking, no, because the computer uses a fixed formula (*algorithm,* in computer-talk) to arrive at a series of numbers that are called *pseudo-random.* Since this series is so long, and the computer generally starts at a different place in the sequence each time, you will rarely find the numbers repeating. You who are advanced programmers will want to know that the number that the RND statement "works on" (the "argument") affects the start point of the sequence. This is called the *seed.* RND(0) will generate a random number that relates to the VIC 20's built-in

clock. This clock begins counting, in 1/60 of a second intervals, from the time the computer is first powered on. If the argument is less than zero (RND(−1), for example), the random number sequence is automatically reseeded. Arguments greater than zero, as in RND(1), will produce the same random number sequence for any given random number seed. The differences actually have little effect in short programs like this one. Key in the following short program and see what happens:

```
10 INPUT "ENTER ARGUMENT :";X
20 R=RND(X)
30 PRINT R;
40 GET A$:IF A$="" GOTO 40
50 GOTO 20
```

Run it a few times, entering different values for X, and watch the sequences. Press any key to see the next random number. Hit RUN/STOP plus RESTORE between runs to ensure that the computer is fully reset.

The "trick" portion of MIND READING is accomplished by adding the value of the number of the object chosen, which will always be between 1 and 8, and is stored in variable NU, to the value that produces the screen border color.

Remember from the last chapter when we produced varying screen borders with a light yellow screen by incrementing from 284 to 255 and POKEing that number into 36879? MIND READING does the same thing, only the exact number is determined by NU. If the number of the object chosen was 1, then NU+SCREEN=248, and a black border with light yellow screen will be produced.

Simple? Yes. Effective? Usually. Don't try this trick too frequently. Audiences can be smarter than they look! You certainly don't want random reactions to your demonstration of mind reading.

```
10 REM ****************
20 REM *              *
30 REM * MIND READING *
40 REM *              *
50 REM ****************
60 SCREEN=36879

65 REM *** Instructions ***
```

PROGRAM LISTING 2. MIND READING

```
70 PRINT"(CLR)(02 DWN)"
80 POKE SCREEN,216
90 PRINT" WHEN THE SAGE IS OUT"
100 PRINT" OF THE ROOM ENTER"
110 PRINT" NAMES OF 8 COMMON"
120 PRINT" OBJECTS IN THIS ROOM."
130 PRINT" YOU MAY THEN CHOOSE"
140 PRINT" ONE & THE SAGE WILL"
150 PRINT" ATTEMPT TO DETERMINE"
160 PRINT" YOUR CHOICE BY ESP."
170 PRINT"(02 DWN)     HIT ANY KEY"
180 PRINT"     TO BEGIN"
190 GET A$:IF A$="" GOTO 190

195 REM *** Enter Names of Objects ***

200 POKE SCREEN,154
210 FOR N=1 TO 8
220 PRINT"(CLR)(02 DWN)"
230 PRINT" ENTER NAME OF OBJECT:"
240 INPUT OBJECT$(N)
250 NEXT N

255 REM *** Computer/Players Choose ***

260 POKE SCREEN,143
270 PRINT"(CLR)(02 DWN)"
280 PRINT" WOULD YOU LIKE:"
290 PRINT"(DWN)  1.) TO CHOOSE OBJECT"
300 PRINT"(DWN)  2.) COMPUTER CHOOSE"
310 PRINT"(02 DWN)    ENTER CHOICE :"
320 GET A$:IF A$="" GOTO 320
330 CH=VAL(A$)
340 IF CH<1 OR CH>2 GOTO 320

345 REM *** Choose Object ***

350 IF CH=2 THEN NU=INT(RND(0)*8)+1:GOTO 420
360 PRINT"(CLR)(02 DWN)"
370 PRINT"ENTER OBJECT TO"
380 PRINT" BE FOUND:"
390 GET A$:IF A$="" GOTO 390
400 NU=VAL(A$)
410 IF NU<1 OR NU>8 GOTO 390

415 REM *** Change Screen ***

420 COLR=247+NU
430 POKE SCREEN,COLR
440 PRINT"(CLR)"
450 PRINT"(RVS)(BLK) (RED) (CYN) (PUR) (GRN) ;
460 PRINT"(RVS)(BLU) (YEL) (BLK) (RED) (CYN) (PUR) ;
470 PRINT"(RVS)(GRN) (BLU) (YEL) (BLK) (RED) (CYN) ;
480 PRINT"(RVS)(PUR) (GRN) (BLU) (YEL) (BLK) (RED) ;
490 PRINT"(RVS)(CYN) (BLK) (OFF)"
```

PROGRAM LISTING 2—CONT. MIND READING

```
500 FOR N=1 TO 8
510 PRINT N;".) ";OBJECT$(N)
520 PRINT
530 NEXT N
540 PRINT"(DWN)  WHAT WAS CHOSEN??(DWN)"
550 PRINT"(RVS)(BLK) (RED) (CYN) (PUR) (GRN) ;
560 PRINT"(RVS)(BLU) (YEL) (BLK) (RED) (CYN) (PUR) ;
570 PRINT"(RVS)(GRN) (BLU) (YEL) (BLK) (RED) (CYN) ;
580 PRINT"(RVS)(PUR) (GRN) (BLU) (YEL) (BLK) (RED) ;
590 PRINT"(RVS)(CYN) (BLK) (OFF)"
600 GET A$:IF A$="" GOTO 540

605 REM *** Reveal Choice ***

610 PRINT"(CLR)(02 DWN)"
620 PRINT" COMPUTER CHOSE "
630 PRINT" ";OBJECT$(NU)
640 GET A$:IF A$="" GOTO 580
650 RUN

#
```

PROGRAM LISTING 2—CONT. MIND READING

CHAPTER 3

AUTO COST

Type: Personal Application
Size: 2100 bytes, for any size VIC 20

One of the secrets of writing computer programs that can be easily understood and debugged is using variable names that have some relation to their function. AUTO COST is a simple program that will help you calculate the true cost of operating your car. And, because it uses long variable names, you'll find it exceptionally easy to modify for special uses.

If you were reading a program and came across the following lines, which would be easier for you to understand?

```
10  C=P-D
or
20  COST=PRICEPAID-DEPRECIATION
```

Both examples are extremes. The first is cryptic, but uses a minimum of memory. The second is much easier to decipher, at a memory penalty. To be accurate, the longer line would not be allowed under VIC 20's BASIC. The first variable name, COST, contains the keyword COS, used to calculate cosine functions. Never mind that you personally never use COS; because it is a reserved word, VIC will not let you use it in a variable. If you type line 20 into the computer, it will return a ?SYNTAX ERROR message and refuse to go any further. Substitute some other variable name for COST, or abbreviate it to CST, and all will work out just fine.

Unfortunately, there are many reserved keywords, most of them short, and they cannot appear anywhere in your variable name. All of the following variables, potentially very useful in business or personal applications programs, are not allowed by the VIC 20:

```
INCOME
TOTAL
VALUE
ORDERS
INVENTORY
MONTHS
```

However, there are thousands of words which CAN be used to make your program logic simpler to follow. There is one more caveat, however, which is especially important to take note of when using variable names. The following line would cause some serious problems in any program:

```
10  PAID=PAYMENT*MTHS
```

As far as the VIC 20 is concerned, PAID and PAYMENT are exactly the same variable. While its BASIC will allow longer variable names, it looks at only the first two. These two are significant, and the others are ignored. The program will see PAID as PA, and PAYMENT also as PA; and it will probably come up with a wrong answer.

AUTO COST uses long variable names wherever possible in asking the operator for information that helps calculate the costs of driving that vehicle. The first questions ask whether the car is leased or purchased, the monthly payment, and how many months the lease has run to date. These two variables, PMT and MTHS (or, PM and MT to the computer), are used to calculate the amount PAID to date.

Those who purchased their car are asked its original PRICE, and the current SELLing price that could be gotten if they were to trade the car in, or sell it outright. These two values are used to calculate net amount PAID in line 310, which is roughly the equivalent of PAID for the lease-owner.

Other facts are also entered into the program: Beginning odometer reading (BEGN — note abbreviation to avoid use of keyword "IN"), current odometer reading (ODOM), and a calculation made, in line 360, to find out how many MILES difference there are between the two.

All the individual costs are added up to produce a final cost per mile. This figure is rounded off using a handy programming trick. Normally, the VIC 20 will produce very accurate decimal numbers with many more decimal places of precision than you need. If a car costs 11.5321871 cents per mile to drive, wouldn't you rather see just 11.53 cents on the screen? One way to format this figure is to take advantage of the INT function to take only the integer portion of the number.

Thus, INT(11.5321871) would return 11. However, you might want more precision than that — say, two extra decimal places. The trick is to multiply the number times the power of ten that equals the number of decimal places of precision desired. Then take the integer portion of that number, and divide by the same power of ten to produce the result.

For example, we could enter the following statement:

```
RESULT=INT(11.5321871*100)
```

In this case, RESULT would equal 1153. Dividing that by 100 (10 to the second power) would produce 11.53, exactly the result we wanted. As a programming shortcut, we would enter:

```
RESULT=INT(11.5321871*100)/100
```

In an actual program, some variable name would be substituted for 11.532187. In AUTO COST, this technique is used to format most of the output, with PRINTTAB(16) used to arrange the figures in a neat column on the screen. Some other BASICS have a function called PRINT USING. The INT function can be used to simulate some of its features, especially where dollars and cents output is desired.

The 11.53 figure could easily be $11.53. However, when using this technique for money results, if the number ends with a zero, it will be presented as 11.5, not 11.50. HINT: change the number to a string, check to see that there are two characters to the right of the decimal point and, if not, add a zero.

```
10 REM *************
20 REM *           *
30 REM * AUTO COST *
40 REM *           *
50 REM *************
```

PROGRAM LISTING 3. AUTO COST

```
60 PRINT (CLR)(DWN)(DWN)"
70 PRINT " IS THIS CAR:"
80 PRINT "   (DWN)(RVS)(RED)L(OFF)(BLU)EASED"
90 PRINT "   (DWN)(RVS)(RED)P(OFF)(BLU)URCHASED"
100 GET A$:IF A$="" GOTO 100
110 IF A$="L" GOTO 140
120 IF A$="P" GOTO 220
130 GOTO 110
140 PRINT (CLR)(02 DWN)"
150 PRINT " MO. LEASE PAYMENT:"
160 INPUT PMT
170 PRINT "(DWN) HOW MANY MONTHS"
180 PRINT " HAS LEASE RUN?"
190 INPUT MTHS
200 PAID=PMT*MTHS
210 GOTO 320
220 PRINT " (DWN) HOW MANY MONTHS"
230 PRINT " HAVE YOU OWNED CAR"
240 INPUT MTHS
250 PRINT "(DWN) COST OF CAR :"
260 PRINT " INCLUDING INTEREST,"
270 PRINT " TAXES, ETC."
280 INPUT PRICE
290 PRINT " (DWN) PRESENT VALUE :"
300 INPUT SELL
310 PAID=PRICE-SELL
320 PRINT "(DWN) ORIGINAL MILEAGE :"
330 INPUT BEGN
340 PRINT " (DWN) PRESENT MILEAGE :"
350 INPUT ODOM
360 MILES=ODOM-BEGN
370 YRS=MTHS/12
380 PRINT "(DWN) YOU HAVE OWNED THIS"
390 PRINT " CAR ";YRS;" YEAR(S)"
400 PRINT "(DWN) ENTER YOUR MPG :"
410 INPUT MPG
420 PRINT (CLR)(02 DWN)"
430 PRINT " WHAT HAS BEEN AVERAGE"
440 PRINT " PRICE OF GAS"
450 INPUT GC
460 GAS=MILES*GC/MPG
470 PRINT "(DWN) YEARLY INSURANCE COST:"
480 INPUT YIC
490 PRINT "(DWN) ANNUAL REGISTRATION:"
500 INPUT PLATES
510 PRINT "(DWN) REPAIRS TO DATE :"
520 INPUT REPAIRS
530 PRINT "(DWN) ACCESSORIES OR REP-"
540 PRINT " PLACEMENT ITEMS SUCH"
550 PRINT " AS TIRES, ETC."
560 INPUT ACCESS
570 PRINT " (DWN) ROUTINE MAINTENANCE"
580 PRINT "  SUCH AS OIL CHANGES:"
590 INPUT RMT
600 TNS=YIC/12*MTHS
```

PROGRAM LISTING 3—CONT. AUTO COST

```
610 PLATES=PLATES/12*MTHS
620 CST=TNS+PLATES+PAID+GAS+REPAIRS
    +ACCESS+RMT
630 CST=INT(CST)
640 PRINT (CLR)(02 DWN)"
650 PRINT " THIS CAR HAS COST"
660 PRINT " YOU $";CST;" TO DATE."
670 CST=INT(CST/MILES*100)
680 PRINT " THIS IS ";CST
690 PRINT " CENTS PER MILE."
700 PRINT"(02 DWN)     (RVS)(RED)HIT ANY
         KEY(OFF)(BLU)"
710 GET A$:IF A$="" GOTO 710
720 PRINT (CLR)(02 DWN)"
730 PRINT "  (RVS)(RED)PER MONTH:(BLU)(OFF)"
740 PAID=INT(PAID/MTHS*100)/100
750 PRINT "(DWN) DEPRECIATION:";
760 PRINT TAB(16);PAID
770 PLATES=INT(PLATES/MTHS*100)/100
780 PRINT "(DWN) REGISTRATION:";
790 PRINT TAB(16);PLATES
800 YIC=INT(YIC/12*100)/100
810 PRINT "(DWN) INSURANCE:";
820 PRINT TAB(16);YIC
830 ACCESS=INT(ACCES/MTHS*100)/100
840 PRINT "(DWN) ACCESSORIES:";
850 PRINT TAB(16);ACCESS
860 GAS=INT(GAS/MTHS*100)/100
870 PRINT "(DWN) GASOLINE:";
880 PRINT TAB(16);GAS
890 RMT=INT(RMT/MTHS*100)/100
900 PRINT "(DWN) MAINTENANCE:";
910 PRINT TAB(16);RMT
920 MILES=INT(MILES/MTHS*100)/100
930 PRINT "(DWN) TRAVELED ";MILES
940 PRINT " MILES PER MONTH."
950 GET A$:IF A$="" GOTO 950

#
```

PROGRAM LISTING 3—CONT. AUTO COST

CHAPTER 4

SPACE COMMAND

Type: Game, one player
Size: 5900 bytes, VIC 20 with any expansion cartridge

Space Command is a science fiction adventure that is suited for fairly young children, especially *Star Wars* fans. The player begins with 50 rebel fighters, 1000 units of fuel, and 1000 photon torpedoes spread among its ships.

At each turn, the player is given the option of flying toward the rebel base, going for fuel, or looking for additional ammunition. If fuel runs out, or the player is attacked by the vicious Quillons when out of ammunition, the game is lost. Entering hyperspace to venture toward the base can be dangerous, as well.

SPACE COMMAND teaches the use of subroutines to accomplish tasks that are carried out over and over throughout a program. Subroutines are a useful tool that allow the programmer to avoid writing the same code repeatedly. They save memory, but can make a program more difficult to understand if they are not labeled logically.

Subroutines are accessed through the "ON . . . variable . . . GOSUB . . . number list" syntax; where the number list is a series of program lines where each of the subroutines resides. For example, you might enter the following program lines:

```
10   INPUT "ENTER YOUR CHOICE (1-5)";CH
20   ON CH GOSUB 100,200,300,400,500
30   GOTO 10
```

```
100  (Get Fuel Subroutine)
. . .
200  (Get Ammo Subroutine)
. . .
300  (Attach Subroutine)
. . .
400  (Go Toward Base Subroutine)
. . .
500  (Flee Subroutine)
. . .
```

If each of these subroutines is clearly labeled with REMarks, the program logic will be possible to follow. In many cases, constructing a program of smaller subroutines can be simpler for the programmer, because it is possible to run each module separately and check the variables going in and coming out to see that they have the desired values.

This program, because it contains so many messages to the player, is somewhat longer than the previous examples. In fact, it is nearly 6000 bytes long and involves some 270-plus non-REM program lines. You'll need any expander cartridge to fit it all in. The Super Expander or 3K Memory Expander cartridges will do just fine. The extra memory of the 8K or 16K cartridge is not needed, but either will work as well.

The instructions to the program are presented at lines 180–380. Then, the initial values of the number of fighter ships, fuel carried, distance from base, and so forth are set in lines 390–400. Each new turn begins at line 420. A counter variable, TURN, is increased by one each time through to keep track of the number of turns required by the player to reach the base.

Between lines 440 and 470, four checks are made to see if, respectively, the amount of fuel remaining (F), number of photon torpedoes (A), distance to go (D), or number of ships (T) is less than one. If any of these conditions occur, control branches to portions of the program intended to deal with them.

Then, the current status is printed to the screen, and the player is offered the choice of getting more fuel, flying toward the rebel base, or searching for an ammunition dump. Each of these options is treated in separate subroutines, located at lines 620, 1030, and 1600; and program control transferred with the ON CH GOTO . . . statement in line 600.

Some of these subroutines have routines of their own. For example, when looking for fuel, a random number between 1 and 5 is chosen (line 690) and used to send control to four separate mini-adventures. These can result in fuel being added, or going away empty handed.

Choosing to head toward the base (the subroutine at line 1030) triggers a jump into hyperspace that is simulated by changing the border and screen to black and printing white "stars" against this field. Then, the screen is restored to indicate a return to normal space, and a random number four or smaller is chosen to determine the outcome. Another random number, U, 500 or smaller, is selected in line 1170 to indicate how far toward, or away from, the rebel base the ship has flown. Heading for ammo produces a similar range of results, triggered by similar random number choices. Depending on the scenarios, the variables storing the number of ships, ammo, distance, and fuel will be added to or subtracted from, as appropriate. Should the player run out of ships, or be attacked when out of ammo, the program branches to one of the "lose" routines which explain the situation. Reaching the rebel base safely (D is less than 1) prompts the computer to bestow accolades and compare the number of turns required with the current RECRD and, if TURNS is less, announce that a new standard has been established. The player can start another game in an attempt to beat that record.

```
10 REM *****************
20 REM *                 *
30 REM * SPACE COMMAND *
40 REM *                 *
50 REM *****************
60 PRINT"(CLR)"
70 RECRD=1000
80 PRINT"    (RVS)(RED)SPACE COMMAND(OFF)(BLU)"
90 PRINT"(DWN)PLEASE ENTER YOUR NAME"
100 INPUT A1$
110 PRINT
120 PRINT"(DWN)INSTRUCTIONS (Y/N) ?"
130 GET A$:IF A$=""GOTO 130
140 IF A$="N" GOTO 390
150 IF A$="Y" GOTO 180
160 GOTO 130

165 REM *** INSTRUCTIONS ***

180 PRINT"(CLR)"
190 PRINT" YOU ARE SETTING OUT"
```

PROGRAM LISTING 4. SPACE COMMAND

```
200 PRINT" FOR THE REBEL BASE"
210 PRINT" WITH 50 FIGHTERS"
220 PRINT" AND 1000 FUEL UNITS."
230 PRINT" YOU ALSO HAVE 1000"
240 PRINT" PHOTON TORPEDOES."
250 PRINT"(DWN) YOU ARE 1000 PARSECS"
260 PRINT" FROM YOUR BASE."
270 PRINT"   ## (RVS)(BLK)BEWARE(OFF)
        (BLU) !! ##"
280 PRINT" THE QUILLONS MAY"
290 PRINT" ATTACK YOU BEFORE"
300 PRINT" YOU ARRIVE. IF YOU"
310 PRINT" RUN OUT OF FUEL,OR"
320 PRINT" ALL YOUR SHIPS ARE"
330 PRINT" DESTROYED, YOU LOSE."
340 PRINT" IF THE ENEMY ATTACKS"
350 PRINT" WHEN YOU ARE OUT OF"
360 PRINT" AMMO, YOU LOSE."
370 PRINT"(DWN)   ==(RVS)(RED)HIT ANY KEY
        (OFF)(BLU)=="
380 GET A$:IF A$="" GOTO 380
390 F=1000:T=50
400 D=1000:A=D

405 REM *** NEW TURN ***

420 PRINT"(CLR)(02 DWN)"
430 TURN=TURN+1
440 IF F<1 THEN 2710
450 IF A<1 THEN 1560
460 IF D<1 THEN 2530
470 IF T<1 THEN 2680
480 V=0:C=0:N=0
490 PRINT" FUEL:";F
500 PRINT" SHIPS:";T
510 PRINT" DISTANCE:";D
520 PRINT" TORPEDOES:";A
530 PRINT"(DWN) DO YOU WANT TO"
540 PRINT"(DWN) 1.) GET FUEL"
550 PRINT" 2.) FLY TOWARD BASE"
560 PRINT" 3.) GET AMMO"
570 GET A$:IF A$="" GOTO 570
580 CH=VAL(A$)
590 IF CH<0 OR CH>3 GOTO 570
600 ON CH GOTO 620,1030,1600

605 REM *** GET FUEL ***

620 PRINT"(CLR)(02 DWN) WE'RE GOING FOR FUEL"
630 PRINT" SCANNERS ARE LOOKING"
640 PRINT" FOR NEAREST REBEL"
650 PRINT" FUEL DEPOT. STAND BY"
660 PRINT " ";A1$;"!(02 DWN)"
670 FOR N=1 TO 2000
680 NEXT N
```

PROGRAM LISTING 4—CONT. SPACE COMMAND

```
690 C=INT(RND(1)*4)+1
700 ON C GOTO 840,770,920,710,710
710 PRINT" YIPES! DEPOT EMPTY!"
720 PRINT" WE STILL HAVE";F
730 PRINT" UNITS OF FUEL."
740 PRINT"(DWN)   ==(RVS)(RED)HIT ANY KEY
        (OFF)(BLU)=="
750 GET A$:IF A$="" GOTO 750
760 GOTO 420
770 F=F+1000
780 PRINT" WE FILLED UP WITH"
790 PRINT" FUEL.  WE NOW HAVE"
800 PRINT" ";F;" UNITS."
810 PRINT"(DWN)   ==(RVS)(RED)HIT ANY KEY
        (OFF)(BLU)=="
820 GET A$:IF A$="" GOTO 750
830 GOTO 420
840 F=F+500
850 PRINT" THE DEPOT COULD ONLY"
860 PRINT" GIVE US HALF A TANK!"
870 PRINT" WE NOW HAVE";F
880 PRINT" UNITS OF FUEL."
890 PRINT"(DWN)   ==(RVS)(RED)HIT ANY KEY(OFF)(BLU)=="
900 GET A$:IF A$="" GOTO 750
910 GOTO 420
920 W=INT(RND(1)*800)
930 F=F+W
940 PRINT" QUILLONS ARE COMING!"
950 PRINT" WE HAD TO LEAVE"
960 PRINT" BEFORE WE COULD FILL"
970 PRINT"  UP.  WE DID GET"
980 PRINT" ";F;" UNITS OF FUEL."
990 PRINT"(DWN)   ==(RVS)(RED)HIT ANY KEY(OFF)(BLU)=="
1000 GET A$:IF A$="" GOTO 750
1010 GOTO 420

1015 REM *** GO BASE ***

1030 PRINT"(CLR)(02 DWN)"
1040 PRINT" WE'RE GOING INTO"
1050 PRINT" HYPERSPACE. HANG ON."
1060 FOR N=1 TO 500:NEXT N
1070 POKE 36879,8
1080 PRINT"(WHI)"
1090 FOR N=1 TO 1000
1100 PRINT":*:";
1110 NEXT N
1120 PRINT"(BLU)"
1130 POKE 36879,27
1140 H=INT(RND(0)*3)+1
1150 IF H=2 THEN 2050
1160 C=INT(RND(0)*4)+1
1170 U=INT(RND(0)*500)+1
1180 ON C GOTO 1260,1330,1420,1190
1190 U=INT(U/10)
```

PROGRAM LISTING 4—CONT. SPACE COMMAND

```
1200 PRINT" YOU WERE CAUGHT"
1210 PRINT" IN A SPACE STORM AND"
1220 PRINT" SENT ";U;" PARSECS"
1230 PRINT" OFF COURSE."
1240 D=D+U
1250 GOTO 1360
1260 U=INT(U*1.5)
1270 D=D-U
1280 PRINT:PRINT"YOU WENT";U;" PARSECS"
1290 PRINT" TOWARD THE BASE."
1300 PRINT" YOU ARE NOW ";D
1310 PRINT" PARSECS AWAY."
1320 GOTO 1360
1330 PRINT" YOU GOT STALLED IN"
1340 PRINT" A TIME WARP, AND GOT"
1350 PRINT" NOWHERE."
1360 F=F-U
1370 PRINT"(DWN) YOU USED UP";U
1380 PRINT" UNITS OF FUEL"
1390 PRINT"(DWN)   ==(RVS)(RED)HIT ANY KEY
          (OFF)(BLU)=="
1400 GET A$:IF A$="" GOTO 750
1410 GOTO 420
1420 PRINT"(CLR)(02 DWN)"
1430 PRINT" YOU ENTERED A"
1440 PRINT" RADIATION AREA.  IT"
1450 V=INT(U/2)
1460 PRINT" CAUSED";V;" PHOTON"
1470 PRINT" TORPEDOES TO FIRE"
1480 PRINT" LUCKY, NONE OF YOUR"
1490 PRINT" SHIPS WERE DESTROYED."
1500 A=A-V
1510 PRINT" YOU HAVE";A;" UNITS"
1520 PRINT" OF AMMO LEFT."
1530 PRINT"(DWN)   ==(RVS)(RED)HIT ANY KEY(OFF)
          (BLU)=="
1540 GET A$:IF A$="" GOTO 1540
1550 GOTO 420
1560 PRINT" OUT OF AMMO!"
1570 PRINT" GO GET MORE!"
1580 GOTO 480

1585 REM *** GET AMMO ***

1600 PRINT"(CLR)(02 DWN)"
1610 PRINT" LOOKING FOR AN AMMO"
1620 PRINT" SUPPLY DEPOT."
1630 FOR N=1 TO 1000
1640 NEXT N
1650 C=INT(RND(1)*6)+1
1660 Q=INT(RND(1)*500)+1
1670 IF C=1 THEN GOTO 1760
1680 IF C=2 THEN GOTO 1830
1690 A=A+Q
1700 PRINT" DEPOT GAVE US";Q
```

PROGRAM LISTING 4—CONT. SPACE COMMAND

```
1710 PRINT" PHOTON TORPEDOES."
1720 PRINT" WE NOW HAVE";A
1730 PRINT"(DWN)   ==(RVS)(RED)HIT ANY KEY(OFF)
          (BLU)=="
1740 GET A$:IF A$="" GOTO 1740
1750 GOTO 1970
1760 PRINT" NO AMMO AVAILABLE"
1770 PRINT" YOU STILL HAVE";A
1780 PRINT" TORPEDOES."
1790 PRINT"(DWN)   ==(RVS)(RED)HIT ANY KEY(OFF)
          (BLU)=="
1800 GET A$:IF A$="" GOTO 1740
1810 GOTO 1970
1820 Q=INT(Q/4)
1830 PRINT" YIPES! YOU WERE"
1840 PRINT" ATTACKED BY QUILLONS"
1850 PRINT" BEFORE YOU COULD GET"
1860 PRINT" TOTHE DEPOT."
1870 IF A<1 THEN GOTO 2630
1880 PRINT"(DWN) YOU USED UP ";Q
1890 PRINT" TORPEDOES TO ESCAPE!"
1900 IF A>0 THEN 1930
1910 PRINT"(DWN) YOU USED UP ALL OF "
1920 PRINT" YOUR AMMO!"
1930 A=A-Q
1940 PRINT"(DWN)   ==(RVS)(RED)HIT ANY KEY(OFF)
          (BLU)=="
1950 GET A$:IF A$="" GOTO 1950
1960 GOTO 1970
1970 PRINT"(DWN) YOU USED UP ";Q
1980 PRINT" UNITS OF FUEL"
1990 PRINT" LOOKING FOR AMMO"
2000 PRINT"(DWN)   ==(RVS)(RED)HIT ANY KEY(OFF)
          (BLU)=="
2010 GET A$:IF A$="" GOTO 2010
2020 F=F-Q
2030 GOTO 420

2035 REM *** QUILLONS ATTACKING ***

2050 PRINT"(CLR)(02 DWN) QUILLONS ATTACKING"
2060 IF A<1 THEN OTO 2670
2070 PRINT" MAKE CHOICE:(DWN)"
2080 PRINT " 1.) ATTACK"
2090 PRINT " 2.) FLEE(DWN)"
2100 GET A$:IF A$="" GOTO 2100
2110 V=VAL(A$)
2120 IF V<1 OR V>2 GOTO 2100
2130 IF V=2 THEN GOTO 2310
2140 PRINT"ATTACKING.."
2150 FOR N=1 TO 2000:NEXT N
2160 FOR N=1 TO 50
2170 POKE 36879,9
2180 POKE 36879,24
2190 NEXT N
```

PROGRAM LISTING 4—CONT. SPACE COMMAND

```
2200 POKE 36879,27
2210 VU=INT(RND(1)*5)+1
2220 PRINT" YOU WON, BUT YOU"
2230 PRINT" LOST";VU;"SHIPS AND"
2240 I=VU*100
2250 PRINT" USED UP";I;"TORPEDOES."
2260 A=A-I
2270 T=T-V
2280 PRINT"(DWN)    ==(RVS)(RED)HIT ANY KEY(OFF)
          (BLU)=="
2290 GET A$:IF A$="" GOTO 2290
2300 GOTO 420
2310 PRINT" ESCAPING.."
2320 FOR N=1 TO 2000:NEXT N
2330 V2=INT(RND(1)*200)+1
2340 PRINT"(DWN) YOU USED UP";V2
2350 PRINT" UNITS OF FUEL, BUT"
2360 J=INT(RND(1)*3)+1
2370 IF J=1 THEN GOTO 2480
2380 PRINT" YOU WERE ATTACKED"
2390 PRINT" ANYWAY.  YOU LOST"
2400 PRINT" ";J;" SHIPS."
2410 PRINT"(DWN) YOU USED UP";V2
2420 PRINT" TORPEDOES."
2430 A=A-V2
2440 PRINT"(DWN)    ==(RVS)(RED)HIT ANY KEY(OFF)
          (BLU)=="
2450 GET A$:IF A$="" GOTO 2450
2460 T=T-J
2470 GOTO 420
2480 PRINT" YOU ESCAPED."
2490 PRINT"(DWN)    ==(RVS)(RED)HIT ANY KEY(OFF)
          (BLU)=="
2500 GET A$:IF A$="" GOTO 2500
2510 GOTO 420

2515 REM *** YOU WIN ***

2530 PRINT" YOU HAVE REACHED"
2540 PRINT" BASE SAFELY WITH"
2550 PRINT" ";T;" SHIPS LEFT!"
2560 PRINT" YOU TOOK ";TURNS;" TURNS."

2565 REM *** NEW RECORD? ***

2580 IF TURNS<RECRD THEN RECRD=TURNS:GOTO 2600
2590 GOTO 2730
2600 PRINT" NEW RECORD!"
2610 GOTO 2730

2615 REM *** YOU LOSE ***

2630 PRINT" QUILLONS ATTACKED"
2640 PRINT" WHEN YOU WERE OUT"
2650 PRINT" OF AMMO. YOU ARE"
```

PROGRAM LISTING 4—CONT. SPACE COMMAND

```
2660 PRINT" DEAD. YOU LOSE!!"
2670 GOTO 2730
2680 PRINT" ALL YOUR SHIPS ARE"
2690 PRINT" DESTROYED. YOU LOSE."
2700 GOTO 2730
2710 PRINT" OUT OF FUEL. YOU ARE"
2720 PRINT" DEAD. YOU LOSE!"
2730 PRINT" (02 DWN)ANOTHER GAME?"
2740 GET A$:IF A$="" GOTO 2740
2750 IF A$="Y" GOTO 390

#
```

PROGRAM LISTING 4—CONT. SPACE COMMAND

CHAPTER 5

ALBUM TIMER

Type: Personal Application
Size: 1300 bytes, for any size VIC 20

Some tasks are easy to perform on a pocket calculator. Other arithmetic functions are better suited to a computer because they require a short program of some type. For example, if you need to know how long all the selections are on a given record album, it's not practical to simply type in the timings on a calculator. Normal math is done with base-10 arithmetic, while time is based on 60 seconds in a minute, and 60 minutes in an hour.

Of course, it is relatively simple to add up all the seconds on a record, multiply times 60, then add in all the whole minutes, divide that by 60 to find out hours, then multiply the remainder times 60 to calculate whole minutes remaining, subtract them, and then multiply the fraction times 60 to determine seconds. Whew!

Better by far, use ALBUM TIMER, which performs all those calculations for you. Add up the timings on a single side, or on a full record, in order to determine what length cassette tape to use. Or, add up timings to find out which records will fit on opposite sides of the same length tape. This program will do both.

It also introduces the flexible way in which the VIC 20 can embed graphics characters in simple print statements. If you wish to print a row of blocks or some other character, just type PRINT, quotation marks, and type in the characters you wish to see. This

program uses the checkerboard block in the VIC 20 character set for decorative effect. This graphics character is produced by holding down the Commodore key and the plus sign simultaneously.

If you'll examine the VIC 20 keyboard, you will see two sets of graphics characters printed on the keyfronts. The left set is summoned by holding down the Commodore key, while the right set is produced by pressing the shift key at the same time. Although either shift key will work, beginners are better off using the one on the left. This procedure makes it easier to remember how to get each character set. Look at the Commodore key and the left shift key in relation to each other. The shift key is on the right, and produces the right character set. The Commodore key is on the left, and is used to produce the left character set. Easier to remember, now?

Another special character, a "ball," produced by entering shift-Q, is used in the menu of this program. Operation is very simple. The user enters minutes and seconds for a selection; these are stored in MI and SE, respectively. Then the program adds these to total minutes (M) and total seconds (S) to provide a subtotal. Before this is displayed on the screen, however, the program runs through that set of steps outlined previously to find out how many whole minutes and seconds have been added. These are printed on the screen, and the user offered the opportunity to hit any key to continue, or one of the four special function keys to stop timing that particular record or side.

Been wondering how to program those special function keys, have you? Well, as described in earlier chapters, pressing any key on the VIC 20 keyboard simply returns a CHR$ code that stands for that particular key. To test this, type in the following program lines.

```
10  GET A$: IF A$ = "" GOTO 10
20  PRINT "KEY PRESSED WAS:"
30  A = ASC(A$)
40  PRINT "CHR$(";A;")"
50  PRINT "WHICH IS :";CHR$(A)
60  GOTO 10
```

This short program will loop at line 10 until a key is pressed. Then, it will tell you the ASCII, or CHR$ code, number of that key and attempt to print the actual CHR$ itself. For example, if the

letter "A" which is also CHR$(65) is pressed, then the variable A in line 30 will have a value of 65. ("ASC" is simply a function to return the CHR$ code of the first character of the string variable inside the parentheses.)

The program will then, in line 40, tell you what the CHR$ code is (it will print "CHR$(65)" in this case) and print that actual CHR$ to the screen. As we learned, some CHR$'s are just control codes. Run the preceding program once, and press RETURN to see what happens. CHR$(13) is the ASCII code for RETURN. Try some other keys, including those requiring a CTRL- or SHF- or CBM-entry to see what happens.

Those mysterious special function keys at the right of your keyboard are nothing more than additional keys with their own CHR$ codes assigned. They do NOTHING other than return that code when pressed. For them actually to have special functions, the programmer must program them into the software. The special function keys' CHR$ codes are:

F1 . . . 133
F3 . . . 134
F5 . . . 135
F7 . . . 136
F2 . . . 137
F4 . . . 138
F6 . . . 139
F8 . . . 140

You will notice that the odd numbered special function keys are produced in unshifted mode, while the shift key must be pressed to get the even number keys. That is why F1,F3,F5,and F7 are consecutive, and F2,F4,F6,and F8 follow. Using the test program, press several of the special function keys to see what happens.

All of the VIC 20 programs using those keys which you might have used or seen before did nothing more than scan the keyboard for those particular CHR$ codes. When seen, control branched to some subroutine or program module that carried out the special function. The keys themselves cannot do functions, and there is no way to "load" them with programs.

Surprised? ALBUM TIMER is the first program in this book to use the special function keys. It is not selective; if the user presses any of the keys, control branches to the "total" portion of the program.

For clarity's sake, the program, in lines 340 and 350, examines A$ to see if it is CHR$(133) or CHR$(134) . . . and so on. It might have been simpler to write:

```
350 IF ASC(A$) < 141 AND ASC(A$) > 132 GOTO 370
```

In fact, you may substitute that line in the program if you wish. However, I used the other style to make it clearer for those applications where specific special function keys carry out specific tasks. We might have typed:

```
360  IF A$ = CHR$(133) GOTO 1000
370  IF A$ = CHR$(134) GOTO 2000
. . .
. . .
```

And so on. Or, after checking to see that it was a special function key that was pressed (as in line 350 of the example), we could have a line like:

```
380 ON VAL(A$) - 132 GOTO 1000,2000,3000,4000 . . ., etc.
```

Special function keys are very useful. We will be using them in later programs, particularly VIC ORGAN, which uses the keys to set musical voices and to trigger the playback sequence.

```
10 REM ***************
20 REM *               *
30 REM * ALBUM TIMER *
40 REM *               *
50 REM ***************
60 FL=1
70 GOTO 90
80 POKE 36879,X:RETURN
90 X=120:GOSUB 80
100 PRINT"(CLR)(02 DWN)      (BLU)(REV)ALBUM
          TIMING"
110 IF FL=1 THEN PRINT"(02 DWN)(DWN)
             (WHI)    COPYRIGHT 1983
             (DWN)         DAVID D. BUSCH"
120 PRINT"(03 DWN)    (BLK) (REV)HIT ANY KEY"
130 GET A$:IF A$=""GOTO 130
140 X=156:GOSUB 80
150 PRINT"(CLR)(BLU)(22 CBM-+)(02 DWN)(BLK)"
160 PRINT"(CLR)(BLU)(22 CBM-+)(02 DWN)(BLK)"
170 PRINT "  (SHF-Q) ENTER SELECTION"
180 PRINT "    TIMINGS.(DWN)"
190 PRINT "  (SHF-Q) HIT ANY SPECIAL"
```

PROGRAM LISTING 5. ALBUM TIMER

```
200 PRINT "     FUNCTION KEY TO "
210 PRINT "     FINISH INPUT.(02 DWN)"
220 PRINT"(BLU)(22 CBM-+)(DWN)(BLK)"
230 N=N+1
240 PRINT"MINUTES FOR #";N;":";
250 INPUT MI
260 PRINT "SECONDS FOR #";N;":";
270 INPUT SE

275 REM *** CALCULATE SUBTOTAL ***

280 M=M+MI:S=S+SE
290 S2=S/60:S3=INT(S2):S1=S2-S3:S4=S1*60
300 M=M+S3:S=S4
310 PRINT"(DWN)(BLU)TOTAL:";M;"MIN";S;"SEC(BLK)"
320 PRINT"(DWN) (REV)(RED)HIT F1-F8 TO STOP."
        :PRINT"(REV)ANY OTHER TO CONTINUE"
330 GET A$:IF A$="" GOTO 330

335 REM *** CHECK FOR FUNCTION KEY ***

340 IF A$=CHR$(133) OR A$=CHR$(134) OR A$=CHR$(135)
    OR A$=CHR$(136) GOTO 370
350 IF A$=CHR$(137) OR A$=CHR$(138) OR A$=CHR$(139)
    OR A$=CHR$(140) GOTO 370
360 GOTO 150
370 X=24:GOSUB 80
380 PRINT"(CLR)(REV)(YEL)(23 SPACES)(OFF)
        (03 DWN)(RED)    (REV)TOTAL TIMING :(OFF)"
390 PRINT"(DWN)   ";M;"MINUTES"
400 PRINT"    ";S;"SECONDS"
410 PRINT"(12 DWN)(YEL)(REV)HIT ANY KEY TO
        RESTART"
420 GET A$:IF A$=""GOTO 420
430 RUN

#
```

PROGRAM LISTING 5—CONT. ALBUM TIMER

CHAPTER 6

COST ESTIMATOR

Type: Home Application
Size: 3000 bytes, for any size VIC 20

Here's another home applications program. It's actually a greatly simplified program that illustrates how professionals are using computers to compile bills of material when planning construction. It will roughly calculate the amount of materials needed to frame and finish a room in your home — such as when dividing a basement or attic area into a playroom or den.

You can use it to do some blue-sky dreaming of that playroom your family has been meaning to add. Go take a few measurements, find out what materials cost, and this program will give you an estimate of how hard the project will hit your bank account.

COST ESTIMATOR assumes that 8-foot-high walls will be built and that ceiling rafters are already in place. Standard 8 x 2 x 4 lumber should be used, studs erected on 16-inch centers, and standard 4 x 8 foot wallboard or paneling purchased. Flooring can be carpet, lineoleum, or either 9- or 12-inch tiles. The latter can be self-stick, or require an adhesive.

That's a lot of restrictions, right? Those used by professionals are a lot more thorough and flexible. They will figure the number of nails and how much dry-wall tape to buy, as well. More and more architectural and engineering firms are using computer-assisted drafting and design tools to streamline their work. Instead of drawing on paper, they will sit down at a com-

puter terminal, usually one that is especially dedicated to design work.

There, they can type in commands, use a digitizer, joystick, or some other tool to position a cursor on a screen, and literally "draw" the design electronically.

A byproduct of this work can be a bill of materials, calculated entirely by the computer. COST ESTIMATOR is an idiot son version of these programs — since all input has to come directly from the user and not from a computer drafting program. We'll look at more CAD simulations in later programs, including FLOOR PLANNER (Chapter 12).

COST ESTIMATOR is similar to AUTO COST in that it consists primarily of a series of INPUT questions asking for data from the operator. The program then uses this information to calculate the number of studs, wall panels, floor covering, adhesive, and so forth.

Because of its limited scope, the program is REALLY strict. The width of the room cannot be greater than the length. You can, however, enter a strange dimension, such as 10 feet, 27 inches. The program will be neither fooled, nor amused, as it converts these dimensions all to inches anyway.

In calculating the number of studs to be used, an extra is allowed per corner per wall. The program is also a little dumb in figuring number of wall panels needed. It will take any leftover dimensions for each wall, and require that partial sheets of that width be used for each. The program won't check to see if the leftovers from one wall might be big enough to make a partial sheet for the next.

If floor tiles are used, the program wants to know if they will be 9- or 12-inch tiles. If not self-stick, you must enter how many square feet a gallon of adhesive will cover, and how much the "sticky stuff" costs. The program will round off the gallons. Home construction is never very precise.

We wouldn't want COST ESTIMATOR to cut things too fine, would we? Who among us has not made an . . . error . . . when cutting wood or paneling? In fact, you would be well advised to add a fudge factor, based on your own particular skill, if using this program in a real situation.

Because current prices can be entered, the program will give you a rough estimate of how much your framing project will cost, as well as how much of each material to buy.

```
10 REM *************** *
20 REM *                *
30 REM * COST ESTIMATOR *
40 REM *                *
50 REM *************** *

55 REM *** INSTRUCTIONS ***

60 PRINT"(CLR)(02 DWN)"
70 PRINT" THIS PROGRAM WILL"
80 PRINT" ROUGHLY FIGURE THE"
90 PRINT" MATERIALS NEEDED TO"
100 PRINT" FRAME AND FINISH A "
110 PRINT" ROOM WITH EIGHT FOOT"
120 PRINT" CEILINGS.  THE PRO-"
130 PRINT" GRAM ASSUMES THAT "
140 PRINT" YOU WILL BE USING "
150 PRINT" 8 FOOT 2 X 4'S AND "
160 PRINT" STANDARD 4 X 8 FOOT"
170 PRINT" WALLBOARD/PANELING."
180 PRINT" TO CALCULATE THE"
190 PRINT" NUMBER OF WALL STUDS"
200 PRINT" 16 INCH CENTERS ARE"
210 PRINT" USED, AND AN EXTRA"
220 PRINT" STUD ALLOWED PER "
230 PRINT" WALL, PER CORNER."
240 PRINT"(DWN)    (RVS)(RED)HIT ANY KEY"
250 GET A$:IF A$="" GOTO 250

255 REM *** ENTER DIMENSIONS ***

260 PRINT"(CLR)(02 DWN)"
270 PRINT" ENTER LENGTH OF"
280 PRINT" ROOM IN FEET, INCHES"
290 INPUT "(02 DWN) FEET :";F
300 INPUT "(02 DWN) INCHES :";I
310 L=F*12+I
320 PRINT"(CLR)(02 DWN)"
330 PRINT" ENTER WIDTH OF"
340 PRINT" ROOM IN FEET, INCHES"
350 INPUT "(02 DWN) FEET :";F
360 INPUT "(02 DWN) INCHES :";I
370 W=F*12+I
380 IF W>L THEN PRINT"(DWN) WIDTH CANNOT
    BE":PRINT" GREATER THAN LENGTH!(DWN)":GOTO 270
390 SI=L*W

395 REM *** ENTER MATERIALS ***

400 PRINT"(CLR)(02 DWN)"
410 PRINT" ENTER UNIT COST FOR"
420 PRINT" 4 X 8 FOOT LUMBER"
430 INPUT TW
440 PRINT"(CLR)(02 DWN)"
450 PRINT" ENTER COST EACH OF"
```

PROGRAM LISTING 6. COST ESTIMATOR

```
460 PRINT" 4 X 8 PANELING OR"
470 INPUT" WALLBOARD :";WB
480 PRINT"(CLR)(02 DWN)"
490 PRINT" FLOOR COVERED WITH :(DWN)"
500 PRINT" 1. FLOOR TILE"
510 PRINT" 2. CARPET"
520 PRINT" 3. LINOLEUM"
530 PRINT"(DWN)  ENTER CHOICE"
540 GET CH$:IF CH$="" GOTO 540
550 FL=VAL(CH$)
560 IF FL<1 OR FL>3 GOTO 530
570 ON FL GOTO 580,780,780
580 PRINT"(CLR)(02 DWN)"
590 PRINT" USING 9-INCH OR"
600 PRINT" 12 INCH FLOOR TILES:"
610 INPUT TYPE$
620 TYPE=VAL(TYPE$)
630 IF TYPE=9 GOTO 650
640 IF TYPE=12 GOTO 650 ELSE GOTO 590
650 SQTLE=TYPE*TYPE
660 PRINT"(CLR)(02 DWN)"
670 PRINT" ENTER COST EACH TILE"
680 INPUT" (USE $0.00)";TC$
690 PRINT"(CLR)(02 DWN)"
700 INPUT" SELF-STICK (Y/N)";AN$
710 PRINT"(CLR)(02 DWN)"
720 IF LEFT$(AN$,1)="Y" GOTO 840
730 PRINT"(CLR)(02 DWN) ENTER COVERAGE OF"
740 PRINT" FEET PER GALLON :"
750 INPUT AD$
760 PRINT"COST PER GALLON :"
770 INPUT AC$:GOTO 820
780 PRINT"(CLR)(02 DWN) ENTER COST PER"
790 INPUT" SQUARE YARD :";CC$
800 GOTO 920
810 PRINT"(CLR)(02 DWN)"

815 REM *** CALCULATE TOTALS ***

820 AD=VAL(AD$)*144
830 AN=SI/AD
840 TN=SI/SQTLE
850 TN=INT(TN+.9)
860 TC=INT(VAL(TC$)*TN)
870 AC=VAL(AC$)
880 AN=INT(AN+.9)
890 AC=AC*AN
900 AC=INT(AC)
910 GOTO 960
920 CN=SI/1296
930 CC=VAL(CC$)*CN
940 CC=INT(CC)
950 CN=INT(CN+.9)
960 PA=L/96
970 PA=INT(PA+.9)*2
```

PROGRAM LISTING 6—CONT. COST ESTIMATOR

```
980 P2=W/96
990 P2=INT(P2+.9)*2
1000 PLATES=PA+P2
1010 W1=L/48
1020 W2=W/48
1030 VRT=W1*2+W2*2+8
1040 VRT=INT(VRT+.9)
1050 SC=VRT*TW
1060 PC=PLATES*TW
1070 LC=INT(SC+PC)
1080 H1=INT(W1)
1090 H2=INT(W2)
1100 L1=L-(H1*48)
1110 L2=W-(H2*48)
1120 PANELS=H1*2+H2*2
1130 WC=PANELS*WB

1135 REM *** PRINT RESULTS ***

1140 PRINT"(CLR)(02 DWN)"
1150 PRINT" MATERIALS";TAB(15);"COST(DWN)"
1160 PRINT" STUDS:";PLATES+VRT;TAB(15);LC
1170 PRINT" SHEETS:";PANELS;TAB(15);WC
1180 IF L1=0 GOTO 1220
1190 PRINT" PLUS 2 PART SHEETS"
1200 PA=2*WB
1210 PRINT L1;" IN. WIDE :";TAB(15);PA
1220 IF L2=0 GOTO 1260
1230 PRINT" PLUS 2 PART SHEETS"
1240 PB=2*WB
1250 PRINT L2;" IN. WIDE :";TAB(15);PB
1260 XC=PA+PB
1270 IF CN=0 OR FL=2 GOTO 1290
1280 PRINT" SQ.YD. LINOLEUM:":PRINTTAB(10)CN;TAB(15)
          ;CC:GOTO 1330
1290 IF CN=0 OR FL=3 GOTO 1310
1300 PRINT" SQ.YD. CARPET:":PRINTTAB(10)CN;TAB(15)
          ;CC:GOTO 1330
1310 PRINT" TILES :";TN;TAB(15);TC
1320 IF AN<>0 THEN PRINT" GAL. AD.:";AN;TAB(15);AC
1330 TT=INT(LC+WC+XC+CC+TC+AC)
1340 PRINT TAB(16)"###"
1350 PRINT"(02 DWN)  TOTAL :";TAB(13)"$ ";TT

#
```

PROGRAM LISTING 6—CONT. COST ESTIMATOR

CHAPTER 7

REACTION TIMER

Type: Game, for up to nine players
Size: 2000 bytes, for any size VIC 20

This game will let two to nine players find out who has the fastest reactions, in a jiffy. It also serves as an introduction to "jiffies" . . . which are the VIC 20's way of keeping track of time.

The object of the game is to strike a given special function key (those light brown keys at the right of the keyboard) as fast as possible, when the name of that key appears unexpectedly on the screen. The computer chooses one key, from F1,F3,F5, and F7, and a delay interval, which can be very short, or an interminable (seemingly) period. When the name of the key is printed to the screen, the player must press that key as quickly as possible. Pressing the wrong key disqualifies him/her. The computer measures how long it takes to press the correct key, keeps track of each player's best time, and displays a winner at the end of the game. Any number of rounds can be selected in advance.

The VIC 20 has an internal "real time clock" which can be accessed by the user. On power-up, this clock is set to zero, and two reserved variables keep track of how much time has elapsed since the computer was turned on. TIME$ will store the number of seconds that have passed, while TI measures the number of "jiffies," which are units of $\frac{1}{60}$ second.

As a test, turn on your computer and type in ?TIME$ very quickly. If you are fast, it may return a number like 000012, showing that no hours, no minutes, and 12 seconds have passed.

Turn the VIC off again and type ?TI as soon as the READY message appears. You will have to be very fast, indeed, to get a number lower than, say, 200.

Both of these variables can be reset from BASIC programs. TIME$ uses a 24-hour clock, so to set the time to one-thirty (1:30) p.m., you would type:

```
TIME$ = "133000"
```

The program can ask for the hours, minutes, and seconds, and put together a string of its own to set the correct time. Later programs in this book, such as BULLETIN BOARD, will show you how to do this. REACTION TIMER is interested only in a more precise timing mechanism, the variable TI. Simply by reading the value of TI when the name of the special function key is printed to the screen; then checking the new value when the actual correct key is pressed; and subtracting; the computer can calculate how many jiffies were required. And, thus, it computes the number of seconds, accurate to 1/60 of a second.

REACTION TIMER also is the first program in this book to make use of sound. Briefly, sound is achieved on the VIC 20 by POKEing a value in a memory location, much as we did to change the screen/border combinations. The VIC has three musical voices, and one "noise" generator. The memory locations which control them are located at 36874, 36875, 36876, and 36877, respectively. POKEing a number to each of these locations will produce a sound, its pitch depending on the number, which can range between 128 and 255.

This tone will continue until it is turned off again by POKEing a zero in that memory location. Usually, a FOR . . . NEXT loop will separate the two POKE statements, to determine the length of the tone. You can use a variable in the loop and change its value during the program run to change the length of the tone. For example:

```
10 PRINT"ENTER LENGTH OF NOTE:"
20 INPUT DELAY
30 POKE 36878,15
40 POKE 36874,201
50 FOR N = 1 TO DELAY
60 NEXT DELAY
70 POKE 36874,0
```

Did you notice line 30? There is an additional memory location, 36878, which determines the volume of the note. Values of 0 (completely off) to 15 (maximum loudness) should be POKEd to turn the voices on, off, or adjust them in loudness. Line 40 will POKE the value 201, which is approximately the note D on the piano, into the lowest pitched voice on the VIC 20. The note will play until the FOR . . . NEXT loop has counted to the variable DELAY, as entered by the user. You can key in this program and try a few values to see what happens.

A great deal more attention will be paid to music on the VIC 20 in Chapter 16, which explains the VIC ORGAN program. In REACTION TIMER, the program begins with a brief sound and light display. This is achieved by a subroutine at lines 90–150, which POKES a value, X, into the memory location that changes the screen/border colors to a new combination, and then plays a note in the highest VIC 20 voice. That routine is itself called several times by a subroutine beginning at line 180. This module repeatedly changes the value of X, and hence the screen/border colors; then it calls the line 90 routine to effect the change and play the musical note. The process happens so fast that the screen seems to flash and the speaker beeps rapidly.

Next, the number of players and rounds can be selected. A FOR . . . NEXT loop of from 1 to ROUND has nested within it a second loop, from 1 to P, which is the number of players. So, each round consists of a turn by each player. The variable P1 is used to track which player is up.

After the screen is cleared, a message warns player number P1 to await the special function key name on the screen. The delay is chosen in line 530, a random number from 1 to 10000. Another FOR . . . NEXT loop counts off this random number G, and then prints the name of the special function key. The actual key selected is chosen in line 560; it can be any of the special function keys from CHR$(133) to CHR$(136). The CHR$ code, which represents the chosen key, is stored in variable CH; the current number of jiffies is captured in variable B.

Then, the VIC 20 repeats line 610 until a key is pressed. At that moment, the number of jiffies in TI is again read (line 620); the difference is calculated; and the key pressed is compared to see if it is the correct one.

If not, the bad news is broken to the player. Otherwise, the time in seconds is calculated (line 740) and compared with the

previous best time to see if it is a record. If the time is the best for that player, the new time is substituted in the array which keeps track of times, P(n).

After the specified number of rounds have been played, all the best scores (stored in an array P(n)) are compared to see which player wins.

```
10 REM ******************
20 REM *                 *
30 REM * REACTION TIMER *
40 REM *                 *
50 REM ******************
60 W=10000
70 BT=1000
80 GOTO 270
90 POKE 36879,X
100 POKE 36878,15
110 POKE 36876,250
120 FOR L=1 TO 10
130 NEXT L
140 POKE 36876,0
150 RETURN
160 GET A$:IF A$=""GOTO 160
170 RETURN
180 FOR N=1 TO 10
190 X=9
200 GOSUB90
210 X=24
220 GOSUB 90
230 NEXT N
240 X=120
250 GOSUB 90
260 RETURN
270 X=248
280 GOSUB 90

285 REM *** START GAME ***

290 PRINT"(CLR)(BLU)(29 CBM-+)";
300 PRINT"(REV)(RED)R(PUR)E(BLK)A(BLU)C
        (RED)T(BLK)I(RED)O(PUR)N(BLK)";
310 PRINT"(BLU)(51 CBM-+)";
320 PRINT"(BLK)(22 CBM-U)"
330 PRINT"(04 DWN)(BLK)   COPYRIGHT 1983"
        :PRINT"   DAVID D. BUSCH(02 DWN)"
340 PRINT"(02 DWN)     (REV)(RED)HIT ANY KEY
        (BLK)(OFF)(02 DWN)"
350 GOSUB160
360 GOSUB180
370 PRINT"(CLR)(03 DWN) HIT SPECIAL FUNCTION"
380 PRINT" KEY F1,F3,F5,OR F7"
390 PRINT" WHEN NUMBER APPEARS"
400 PRINT" ON SCREEN.  SPEED":PRINT" COUNTS!"
```

PROGRAM LISTING 7. REACTION TIMER

```
410 PRINT"(02 DWN) HOW MANY PLAYERS";
420 INPUT P
430 PRINT"(DWN) HOW MANY ROUNDS";:INPUT ROUND
440 FOR N=1 TO P
450 P(N)=1000
460 NEXT N
470 PRINT"(04 DWN)    (REV)(BLU)HIT ANY KEY"
        :PRINT"(OFF)    (REV)WHEN READY "
480 GOSUB160

485 REM *** START GAME ***

490 FOR Q=1 TO ROUND
500 FOR P1=1 TO P
510 PRINT"(CLR)": GOSUB 180
520 PRINT"(DWN)   (RED)(REV)GET SET PLAYER";P1
530 G=RND(1)*10000

535 REM *** WAIT RANDOM TIME ***

540 FOR N=1 TO G
550 NEXT N

555 REM *** SELECT KEY ***

560 K=INT(RND(1)*4)+1
570 F=K*2-1
580 CH=132+K
590 B=TI
600 PRINT"(04 DWN)";TAB(8)"F";F
610 GET A$:IF A$=""GOTO 610
620 E=TI
630 DF=E-B

635 REM *** SEE IF CHOICE OK ***

640 IF A$=CHR$(CH)GOTO 720
650 PRINT"(CLR)":GOSUB180
660 PRINT"(04 DWN)     SORRY,SPORT"
670 PRINT"      WRONG KEY!"
680 PRINT"(02 DWN)      (REV)(RED)HIT RETURN"
690 PRINT"(OFF)     (REV)TO TRY AGAIN"
700 INPUT G$
710 GOTO 820
720 PRINT"(CLR)":GOSUB180
730 PRINT"(04 DWN)       (REV)(BLK)RIGHT!!!!(OFF)"
740 T=DF/60
750 PRINT"(02 DWN) YOU DID IT IN"
760 PRINT" ONLY ";T:PRINT" SECONDS!"
770 IF DF<BT THEN PRINT"(02 DWN)THAT IS A NEW
    RECORD":BT=T
780 IF DF<P(P1)THEN P(P1)=DF
790 PRINT"(02 DWN)      (REV)(RED)HIT RETURN"
800 PRINT"(OFF)     (REV)TO TRY AGAIN"
810 INPUT G$
```

PROGRAM LISTING 7—CONT. REACTION TIMER

```
820 NEXT P1
830 NEXT Q

835 REM *** GAME OVER ***

840 PRINT"(CLR)":GOSUB 180
850 PRINT"     (REV)(BLK)GAME OVER(DWN)"
860 PRINT TAB(12)"BEST TIME(OFF)(DWN)"
870 FOR N=1 TO P
880 PRINT"PLAYER ";N;TAB(8)P(N)/60
890 IF P(N)<W THEN WP=N:W=P(N)
900 NEXT N
910 PRINT"(DWN) PLAYER ";WP;" WINS."
920 PRINT"(DWN)   PLAY AGAIN?";
930 GOSUB 160
940 IF A$="Y" THEN RUN

#
```

PROGRAM LISTING 7—CONT. REACTION TIMER

CHAPTER 8

STOCK MARKET

Type: Game, one player
Size: 3300 bytes, for any size VIC 20

Remember, those evil, tightly packed, difficult to read programs I warned you about? Well, this is one of them. STOCK MARKET is about the longest program that can possibly be crammed into a stock 5K VIC 20 computer. Multistatement lines, the use of the fewest possible spaces, and other packing techniques are necessary to run the program in an unexpanded VIC 20.

STOCK MARKET is a stock exchange program that will let you buy various stocks, sell them, and make profits, or take losses. You can even sell more stock than you have (this is known as selling short) or spend more money than you have (this is known as an error in judgement). Margin accounts are not implemented in this game, so exceeding your funds actually just provides you with a deficit balance which should be made up at some point in the game.

Most of the VIC 20 tools we have discussed so far are used in this program. The player is alerted to changes in status by a beep from the computer, produced by a routine contained entirely within line 70. The color combinations of the screen are altered by a routine in line 160. Various random effects on the stock market are selected using the RND function discussed earlier.

Each round begins with the display of the "big board," which includes the name of the nine stocks (stored in string array

A$(n)), the Opening price (A(n)), the Closing price (or new value:NV), and the amount of change.

The player is offered the opportunity to Buy, Sell, or Do Nothing each round. When Buying, a new board is displayed, which shows all the stocks and number of shares owned. The player can choose a number of the stock to sell, and specify how many shares to sell. It is possible to sell shares not owned, as this is an allowable procedure on the real stock market. In this case, the number of shares owned will subsequently be displayed as a minus (negative) number.

Selling short in this manner is a dangerous procedure. In normal sales, the risk is limited. The buyer can only lose his/her investment if the stock should happen to go down to zero. But, potential profits are theoretically limitless, as stocks have no practical ceiling price.

However, when selling short, the player chooses to sell stock he does not have at a given price, in hopes of buying it back later at a lower price. The profits have a ceiling — the difference between the selling price and 1, the lowest the stock can go. However, potential losses are unlimited, because the stock could just keep going up.

The program changes the values of the stocks each round, according to the state of the market. Various world events take place, such as strikes, bank failures, and wars. Some of these cause a bull market, in which more stocks rise than fall. Others result in a bear market, during which more stocks go down than go up. In neither case is the movement completely one-sided. Some stocks may buck the trend during a bear market and go up. Others may go down during a bull market. RND numbers chosen by the VIC 20 take care of these factors automatically.

STOCK MARKET isn't a realistic simulation of Wall Street. However, it can be useful in teaching various concepts of trading and showing how some choices can affect some investments. It was intended just for fun, and has no fixed number of rounds, or goals. Quit when you like — or shoot for beating your own previous best investment record.

```
10 REM ***************
20 REM *             *
30 REM * STOCK MARKET *
40 REM *             *
50 REM ***************
60 F=3000:GOTO170
70 POKE36878,15:POKE36876,220
   :FORV=1TO5:NEXTV:POKE36876,0:RETURN
80 DATA 1 ATT,2 IBM,3 CBS,4 TRW,5 NCR,
        6 GAF,7 GM ,8 MCA,9 UAL
90 POKE36879,X:RETURN
100 DATA FED TIGHTENS CREDIT,WAR IN MIDEAST,
    PRIME RISES,DOW DROPS,HEAVY TRADING
110 DATA AUTO WORKERS STRIKE, DOLLAR DOWN,
    GOLD PRICE DROPS,LIVING COST UP
120 DATA MAJOR BANK FALLS,MONEY SUPPLY UP,
    PRIME RATE DOWN,DOW RISES SHARPLY
140 DATA STRIKE SETTLED,GOLD PRICES UP,
    INFLATION DROPS
150 DATA DOLLAR GAINS,NASDAQ FAVORABLE,
    PROFIT TAKING,TRADING ACTIVE
160 POKE36879,X:RETURN
170 PRINT"(CLR)";:X=14:GOSUB160
180 PRINT"(RVS)(BLK)(22 CBM-+)"
190 PRINT"(03 DWN)(OFF)     (OFF)(RVS)(RED)
    STOCK MARKET(OFF)(03 DWN)"
200 PRINT"    COPYRIGHT 1983":PRINT
    "    DAVID D. BUSCH(02 DWN)"
210 PRINT"(07 DWN)(22 CBM-+)";:GOSUB70
    :GOSUB1030
220 X=9:GOSUB160
230 PRINT"(CLR)":X=239:GOSUB160
240 BA=5000:FORN=1TO9:READA$(N):NEXTN

245 REM *** READ DATA ***

250 FOR N=1TO10:READBEAR$(N):NEXTN
    :FORN=1TO10:READBULL$(N):NEXTN
260 FORN=1TO9:R=RND(1)*100
    :A(N)=INT(R)*5+100:NEXTN
270 GOSUB 800
280 FORN=1TO9:B=INT(B(N)):IFB=0THEN300
290 D(N)=(INT(RND(1)*20)*-1):GOTO310
300 D(N)=(INT(RND(1)*20))
310 NEXT N

315 REM *** DISPLAY BOARD ***

320 GOSUB70:PRINT"(CLR)"
330 PRINT"(BLK)NAME(WHI)(SHF-Q)(BLK)
    OPEN(WHI)(SHF-Q)(BLK)CLOSE(WHI)
    (SHF-Q)(BLK)CHANGE"
340 PRINT"######################"
350 FOR N=1TO9:NV=A(N)+D(N)
360 IFNV<2THENNV=1
```

PROGRAM LISTING 8. STOCK MARKET

```
370 IFNV=1THEND(N)=-A(N)+1
380 PRINTA$(N);TAB(5);A(N);TAB(10)
    ;NV;TAB(15);"";
390 IFABS(D(N))<99THENPRINT" ";:GOTO 410
400 IFABS(D(N))<9THENPRINT"  ";
410 IFD(N)>1THENPRINT"+";:GOTO 440
420 IFD(N)<>0THENPRINT"-";:GOTO440
430 PRINT" ";
440 PRINTABS(D(N)):A(N)=NV:NEXTN
450 PRINT"(DWN) BALANCE $";BAL
460 PRINT"(DWN) WOULD YOU LIKE TO"
470 PRINT"(DWN) (RVS)(RED)B(BLU)(OFF)UY
    (RVS)(RED)S(OFF)(BLU)ELL    (RVS)
    (RED)D(OFF)(BLU)O NOTHING
    (BLK)(OFF)":GOSUB1030
480 IF A$="B"GOTO 520
490 IF A$="S"GOTO 710
500 IF A$="D"GOTO 270
510 GOSUB1030:GOTO480

515 REM *** BUY STOCK ***

520 GOSUB70:PRINT"(DWN)"
530 GOSUB 680
540 PRINT"(02 DWN) ENTER # OF STOCK"
550 GETA$:IFA$=""GOTO550
560 GOSUB70:S=VAL(A$):IFS<1GOTO550
570 INPUT"(DWN)ENTER $ INVESTED(DWN) :";D:GOSUB70
580 N1=D/A(S):IFN1<1THENPRINT"(CLR)(DWN)THAT WON'T BUY A
SHARE":FORQ=1TO1000:NEXTQ:GOTO270
590 PRINT"(CLR)(03 DWN) $";D:PRINT" WILL
    PURCHASE"
600 N1=INT(N1):PRINT"  ";N1;" SHARES."
610 PR=A(S)*N1:HO(S)=INT(HO(S)+N1):BAL=BAL-PR
620 IFBAL<1THENPRINT"(DWN) YOU HAVE SPENT MORE
    ":PRINT" MONEY THAN YOU HAVE"
630 FORN=1TO500:NEXTN:PRINT"(DWN) DO YOU WANT TO
    BUY":PRINT" MORE STOCK (Y/N)?":GOSUB1030
640 IFA$="Y"GOTO670
650 IFA$="N"GOTO270
660 GOSUB1030:GOTO640
670 GOSUB680:GOTO540
680 PRINT"(CLR)(02 DWN) BALANCE $";BA
690 GOSUB70:PRINT"(DWN) NAME PRICE  OWNED(DWN)"
700 FORN=1TO9:PRINTA$(N);TAB(7);
    A(N);TAB(12)HO(N):NEXTN:PRINT"(DWN)":RETURN

705 REM *** SELL STOCK ***

710 GOSUB 680
720 PRINT" STOCK TO BE SOLD"
730 GETA$:IFA$=""GOTO730
740 GOSUB70:SB=VAL(A$):IFSB<1GOTO730
750 PRINT" ENTER # SHARES SOLD":
    INPUTN2:GOSUB70:PR=A(SB)*N2:BA=BA+PR
```

PROGRAM LISTING 8—CONT. STOCK MARKET

```
760 HO(SB)=HO(SB)-N2:PRINT" WANT TO SELL":
    PRINT" MORE SHARES (Y/N)?":GOSUB1030
770 IFA$="Y"GOTO710
780 IFA$="N"GOTO270
790 GOSUB1030:GOTO770

795 REM *** MARKET CHANGES ***

800 M=INT(RND(1)*4)+1:Y=INT(RND(1)*10)+1
810 ONMGOTO820,870,920,970
820 PRINT"(CLR)(02 DWN)  ";BULL$(Y);
    ".(DWN)":PRINT"  BULL MARKET.(DWN)":
    PRINT" MOST STOCKS TO RISE."
830 FOR N=1 TO 9:E=RND(1)*10:
    IFE>8THENB(N)=1:GOTO850
840 B(N)=0
850 NEXTN
860 FOR N=1TOF:NEXTN:PRINT"(CLR)":RETURN
870 PRINT"(CLR)(02 DWN)  ";BEAR$(Y);
    ".(DWN)":PRINT"  BEAR MARKET.(DWN)":PRINT"
    MOST STOCKS TO FALL."
880 FOR N=1 TO 9:E=RND(1)*10:IFE<8
    THENB(N)=1:GOTO 900
890 B(N)=0
900 NEXTN
910 FOR N=1TOF:NEXTN:PRINT"(CLR)":RETURN
920 PRINT"(CLR)(02 DWN)   ";BULL$(Y);
    ".(DWN)":PRINT"   MARKET VARIABLE."
930 FOR N=1 TO 9:E=RND(1)*10:IFE<5
    THENB(N)=1:GOTO 950
940 B(N)=0
950 NEXTN
960 FOR N=1TOF:NEXTN:PRINT"(CLR)":RETURN
970 PRINT"(CLR)(02 DWN)   ";BEAR$(Y);
    ".(DWN)":PRINT"    MARKET MIXED."
980 FOR N=1TOF:NEXTN:PRINT"(CLR)":RETURN
990 FOR N=1 TO 9:E=RND(1)*10:IFE<5
    THENB(N)=1:GOTO 1010
1000 B(N)=0
1010 NEXTN
1020 FOR N=1TOF:NEXTN:PRINT"(CLR)":RETURN
1030 GETA$:IFA$=""GOTO1030
1040 RETURN

#
```

PROGRAM LISTING 8—CONT. STOCK MARKET

CHAPTER 9

BULLETIN BOARD

Type: Home Application
Size: 3000 bytes, for any size VIC 20

Angry at your lazy VIC 20 for loafing between sessions of playing games or programming? Maybe you're looking for something that will keep the microchips warmed up and ready for action. Try BULLETIN BOARD. It turns your favorite microcomputer into a home message center that can take notes for you and pass them along to the appropriate member of your family.

Those who communicate with other computer owners through The Source, CompuServe, or various group systems find that such community bulletin boards are very useful. Messages can be posted for later pickup, or sent as more general "EVERYBODY!" broadsides as public service announcements.

BULLETIN BOARD applies the same concept on a smaller scale to your VIC 20. Members of your household or organization can glance at the video screen throughout the day, post messages for others, or retrieve their own.

This program makes more use of string arrays — that handy little series of memory "boxes" which can store a series of characters — as well as our first application of the real time clock to access the real time. BULLETIN BOARD stores up to 50 messages in an array named MESS$(n), and the names of the sender and addressee in two similar arrays, FR$(n) and AD$(n). Because the VIC 20 allocates memory space for these arrays, the program really requires quite a bit more memory than it actually occupies.

The program itself is about 2200 bytes long. But, once the string arrays have been DIMensioned, nearly 3000 bytes have been accounted for. When used with a VIC 20 with memory expansion, the program can be modified to accept more than 50 messages.

The actual message storage and retrieval is the simplest part of the program. A counter, NU, keeps track of the number of messages. When a new message is input, NU is incremented by one (line 950), and the incoming message stored in the next location of the array MESS$(n). When the note is deposited in MESS$(NU), the name of the sender is placed in FR$(NU), while the name of the addressee is stored in AD$(NU). It's as simple as that!

To make things interesting, the current time is added onto the message, so that the recipient can tell when the note was sent. Adding TI$ to MESS$(NU) is simple enough. It is more complicated to set the correct time when the program is first RUN, however.

The user enters the hour, which is stored in HR$, and the current minute, in MN$ (lines 370–430). Each of these is compared to see if it is smaller than 10, in which case a leading zero is added. This makes 9:02 a.m. appear as 090200, which is how the VIC 20 expects to see it. Line 490 transfers this value of T$ to TI$, thus setting the real time clock correctly. The program then appends TI$ to the message at any time it wishes.

Another application for the real time clock is the "message waiting" beeper included with this program. Once activated, the program loops through a GET A$ routine (lines 620–650) that not only checks for keyboard input, but compares the current value of TI$ with two target strings. If the rightmost two characters of TI$ equal "00" or "30," then the real time clock is on the minute or half minute. In such cases, control jumps to a subroutine at line 110, which obligingly beeps.

If no beep was desired, then BFLAG will equal 0, and the time-checking routine will not be accessed each time through. Though the module seems complicated, the VIC 20 is able to process each program line many times each second.

If a key is pressed and it is E (for Enter new message) or R (for retrieve), control goes to respective routines to do just that. However, pressing R will be ignored if there are, in fact, no messages waiting. Clever, these VIC 20's!

At the "Enter" module, NU is incremented, and the user allowed to enter a line or two of message. The choice of turning the beep on is also offered.

The "Retrieval" module will display several screensful of messages, allow keying in one choice, or going back to the previous screen. Ten messages at a time (each of them displayed in order, from the array MESS$(n) along with the sender's name) are shown.

How does the computer know to show just ten messages? Get ready for another programming trick. BULLETIN BOARD cycles through a FOR . . . NEXT loop from 1 to NU, which is the number of messages. In line 740, the loop counter, N, is always checked to see if it can be evenly divided by 10. If so, then control branches to a subroutine that displays the "Hit M for More, P for Previous, or Enter number" message. That routine accepts an answer and does as directed.

In many programs, you will want to see if a number can be divided evenly by another. To do that, we need to see if the first number, when divided by the target number, yields the same result as when the integer of that number is divided by the target. The following example may clarify a bit:

```
10  INPUT N
20  IF N/10 = INT(N/10) PRINT "EVENLY DIVISIBLE"
30  GOTO 10
```

If N = 18, then N/10 will equal 1.8, but will NOT equal INT(N/10), which will be 1. However, if N were 20, then N/10 would equal 2, and INT(N/10) would also equal 2. You can substitute any desired number for 10 and reach the same result.

```
10 REM ******************
20 REM *                 *
30 REM * BULLETIN BOARD *
40 REM *                 *
50 REM ******************
60 GOTO170
70 POKE36879,X:RETURN
80 PRINT"(CLR)(02 DWN)":RETURN
90 GETA$:IFA$=""GOTO90
100 RETURN
110 POKE36878,15
120 POKE 36876,250
130 FOR N=1 TO 500
```

PROGRAM LISTING 9. BULLETIN BOARD

```
140 NEXT N
150 POKE 36876,0
160 RETURN
170 DIM MES$(50),AD$(50),FR$(50),T$(50)
180 X=106
190 GOSUB 70
200 PRINT"(WHI)";
210 GOSUB 80
220 X=106
230 GOSUB 70

235 REM *** START PROGRAM ***

240 PRINT"     (RVS)HOME BULLETIN"
250 PRINT"(OFF)          (RVS)BOARD(OFF)"
260 PRINT"(03 DWN)(BLK)   COPYRIGHT 1983"
270 PRINT"   BY DAVID D. BUSCH"
280 PRINT"(05 DWN)       (RVS)(GRN)HIT ANY KEY
    (OFF)(WHI)"
290 GOSUB 110
300 GOSUB 90
310 GOSUB 80
320 X=14
330 GOSUB 70
340 PRINT"       (YEL)(RVS)ENTER TIME(OFF)(WHI)
    (02 DWN)"
350 INPUT"  CURRENT HOUR";HR$
360 GOSUB 110
370 HR=VAL(HR$)
380 IF HR>23 GOTO 350
390 IF HR<1 GOTO 350
400 IF HR<10 THEN HR$="0"+RIGHT$(STR$(HR),1)
410 PRINT"(02 DWN)  CURRENT MINUTES";
420 INPUT MN$
430 MN=VAL(MN$)
440 GOSUB 110
450 IF MN>59 GOTO 410
460 IF MN<1 THEN MN$="00":GOTO 480
470 IF MN<10 THEN MN$="0"+RIGHT$(STR$(MN),1)
480 T$=HR$+MN$+"00"
490 TI$=T$
500 X=12:GOSUB70
510 GOSUB 70

515 REM *** WAIT FOR ACTIVITY ***

520 GOSUB 80
530 GOSUB 110
540 PRINT"(22 CBM-+)"
550 PRINT"(02 DWN)  HOME BULLETIN BOARD(02 DWN)"
560 PRINT"(RED)(BLU) HIT (RVS)(RED)E(BLU)
    (OFF)NTER MESSAGES(DWN)"
570 PRINT"     (RVS)(RED)R(OFF)(BLU)
    ETRIEVE MESSAGES"
580 PRINT"(WHI)(02 DWN)(22 CBM-+)"
    :IFNM<1GOTO620
```

PROGRAM LISTING 9—CONT. BULLETIN BOARD

```
590 PRINT"(GRN)(02 DWN) ";NM;
600 PRINT"MESSAGE";:IF NM>1 THEN PRINT"S";
610 PRINT" WAITING(WHI)"
620 IF BFLAG=0 GOTO 650
630 IF RIGHT$(TI$,2)="00"THEN GOSUB 110
640 IF RIGHT$(TI$,2)="30" THEN GOSUB 110
650 GET A$:IF A$=""GOTO620
660 IF A$="E"GOTO 950
670 IF NM=0 GOTO 650
680 IF A$="R" GOTO 700
690 GOTO 650

695 REM *** RETRIEVE ***

700 GOSUB 80
710 PRINTTAB(4)"(YEL)TO";TAB(13)
    "FROM(WHI)(DWN)"
720 FOR N=1 TO NU
730 IF MESS$(N)<>""THEN PRINT N;
    TAB(4)AD$(N);TAB(13)FR$(N)
740 IF INT(N/10)=N/10 THEN GOSUB 900
750 NEXT N
760 PRINT"(DWN)(YEL) ENTER NO. OR 'P'
    FOR":PRINT" PREVIOUS LIST(WHI)"
770 INPUT A$:GOSUB 80
780 A=VAL(A$)
790 IF A>NU THEN GOTO 520
800 IF A>0 GOTO 820
810 GOTO 700
820 PRINT"(CLR)(DWN)"
830 PRINT"(GRN)FROM:";TAB(10)FR$(A)
840 PRINT"(DWN) TO:";TAB(10)AD$(A)
850 PRINT"(DWN)TIME :";TAB(10)T$(A)
860 PRINT"(DWN)(WHI)";MESS$(A)
870 PRINT"(DWN)      (RVS)(YEL)HIT ANY KEY
    (WHI)(OFF)"
880 GOSUB 90
890 GOTO 520
900 PRINT"(DWN)(YEL)ENTER NO. OR 'M' FOR
    MORE(WHI)"
910 INPUT A$:GOSUB 80
920 A=VAL(A$)
930 IF A$="M"THEN GOSUB80:RETURN
940 GOTO 820

945 REM *** ENTER MESSAGES ***

950 NU=NU+1
960 GOSUB 80
970 PRINT"(GRN) MESSAGE TO:(DWN)(WHI)"
    :INPUT AD$(NU)
980 PRINT"(DWN)(GRN) MESSAGE FROM:(DWN)
    (WHI)":INPUT FR$(NU)
990 GOSUB 80
1000 PRINT"(DWN)(GRN) ENTER YOUR MESSAGE:
     (WHI)":INPUT MESS$(NU)
```

PROGRAM LISTING 9—CONT. BULLETIN BOARD

```
1010 MESS$(NU)="  "+MESS$(NU)
1020 GOSUB 80
1030 PRINT" DO YOU WANT A SOUND(DWN)   SIGNAL
     TO BEEP  ONCE(DWN)  EACH 30 SECONDS?"
1040 GOSUB 90
1050 IF A$="Y"THEN BFLAG=1:GOTO 1070
1060 IF A$<>"N"GOTO 1040
1070 T$=TI$
1080 HR$=LEFT$(T$,2):MN$=MID$(T$,3,2)
1090 IF LEFT$(HR$,1)="0"THEN HR$=RIGHT$(HR$,1)
1100 T$(NU)=HR$+":"+MN$
1110 NM=NM+1
1120 GOTO 520

#
```

PROGRAM LISTING 9—CONT. BULLETIN BOARD

CHAPTER 10

KITCHEN TIMER

Type: Home Application
Size: 1800 bytes, for any size VIC 20

Because the VIC 20 has a built-in clock, it can do more than just tell time. The computer can also be used to time certain events in the real world. Since most of us spend some time in the kitchen cooking, herewith KITCHEN TIMER, which can be used to prompt you to take your favorite recipe out of the oven after it has been suitably burned to a crisp.

This program takes the clock-setting procedure one step further. In BULLETIN BOARD, it was necessary to input the time in 24-hour form. The user has to know to enter 13 instead of 1 when it is 1:00 p.m. There were no error checks in the program to watch for this, and some conceivably humorous results can happen. (Hi Mom! Just got home! 1:00).

KITCHEN TIMER is smarter than that. If the user enters a time that is larger than 13 hours, the program assumes that a smart person is operating the keyboard, and goes on to ask for the number of minutes. If a number 12 or smaller is entered, the VIC 20 assumes the worst, and asks for clarification. Did you mean a.m. or p.m., please?

When the VIC is satisfied, it puts together a T$ similar to the method used in BULLETIN BOARD, and sets the correct time. Simple enough so far. However, a timer has to keep track of the elapse of time to be of any use. So, the program will ask you how long a period to be timed, in hours and minutes, as well as

whether or not the user wants to be prompted to baste, turn, or stir, and in what interval.

The VIC 20 commences timing, and continually compares the current TI$ with a target time, calculated in a routine at lines 540–640. What the computer does is take the starting time, extract the current hours and minutes, and turn them into numeric values (lines 540–550). Then a "finish hour" and "finish minutes" are figured by adding the current hour and minutes to the amount of time to be ticked off. If the finish minutes are greater than 60, then an additional hour is added to the finish hour. These numbers are used to produce the target string of characters, which is continually compared with TI$ to see if it is time to sound the alarm.

What about the basting/turning signal? Variable G, which is the number of current minutes, is divided by the interval chosen (IV) to see if it can be divided evenly. This is a further application of the screen display routine in BULLETIN BOARD, in which the program looks to see if the number of lines displayed can be evenly divided by 10. In this case, the divisor is the interval, and every time the current minutes can be divided by the interval evenly, control goes to the signaling routine.

Two types of signals are used. One tone and screen signal tells that turning or basting is necessary. Another is used to report that the end of the entire timing cycle has been reached.

The sound, as you know by now, is produced by POKing a number into one of the VIC 20's sound registers. However, we use a slightly different technique to place the elapsing time on the screen. You may have noticed that various effects can be done by POKing different values into memory locations. Well, it is easy to POKE to the screen, as well.

The VIC 20's screen is "memory mapped." That is, an array of memory 506 bytes long keeps track of what characters appear on the screen. You might think of this array as a chart with 22 columns across and 23 rows down. These correspond to the 22 characters that can be printed across the VIC 20 screen, in 23 rows down the side. Actually, in memory, the "boxes" are continuous from the starting memory location to the end, 506 bytes later.

On power-up, most of these boxes contain 32, which is the CHR$ code for a space. Some will have other numbers, standing for the CHR$ codes of the letters, including READY, printed on

the screen. The contents of the video memory boxes are constantly updated as the screen changes. If a letter "A" were to be printed in the upper left-hand corner of the screen and you were to PEEK at box number 1, you would find 65, the CHR$ code for "A" there. Conversely, by POKing a 65 to that memory location, an "A" would appear in the upper left-hand corner.

Confused so far? There's more. The VIC 20 actually has TWO memory maps. The second one, also 506 bytes long, keeps track of what color is being displayed in that particular box. This second map is located in another portion of memory, but has the same number of boxes. To change the color in that box, we POKE a number corresponding to one LESS than the color code desired. That is, to change Box no. 1 to black, we would poke 0 in color memory location number 1. To change it to white, we would poke a 1, and so forth.

Unfortunately, PEEKing those memory locations does not return the expected 0,1,2,3, etc. The reason for this will be discussed later but, briefly, the reason is that the color memory does not use the full 8 bits of each memory location's 8-bit byte. Instead, only 4 bits, or a "nybble," are used. However, PEEKing returns a decimal equivalent of the full 8 bits, and thus will not tell us whether the nybble was a 1,2, or some other number.

Luckily, most programs involve POKing a given color register to change the color, and do not ask us to figure out what color was already there. Whew.

In all programs in this book which POKE to either the color memory or the character memory, two variables, CSCREEN (for color screen), and CHAR (for character screen) are used to represent the starting values for those particular arrays. In the past, we have used real numbers to indicate these memory locations — 36879 for the screen/border value, for example.

It's not possible to do that with the color memory or character memory, however, because these two positions MOVE, depending on how much memory the VIC 20 has. The location is different with a 5K VIC than with a VIC 20 with 8K or 16K expansion cartridges. That is why many programs will run only on an unexpanded VIC 20. These may use POKEs to change screen location colors, or to put particular values on the screen. POKEs that work with the plain-vanilla VIC don't work on an expanded model, unless the changing addresses are taken into account.

KITCHEN TIMER uses a pair of lines, 70 and 80, which examine the memory available, calculate exactly where color and screen memory start, and place these values in the variables CSCREEN and CHAR. Instead of POKing a specific address, the program POKEs to CHAR+1 to place a character at position number one, or POKEs CSCREEN+1 to change the color at the screen location.

It is often necessary to POKE to both memory maps to make a character appear on the screen. Typing POKE CHAR+2,81 will indeed place a "ball" character (CHR$(81)) at the second position on the screen. However, it will be white, and invisible against the white screen, unless you have modified the screen/border colors. By following that POKE with POKE CSCREEN+2,0, the ball will be black and, therefore, visible.

```
10 REM *****************
20 REM *                 *
30 REM * KITCHEN TIMER *
40 REM *                 *
50 REM *****************
60 POKE 36876,191
70 CSCREEN=37888+4*(PEEK(36866)AND128)
80 CHAR=4*(PEEK(36866)AND128)+64*
   (PEEK(36869)AND120)

85 REM *** SET TIME ***

90 PRINT "(CLR)(DWN) SET CURRENT TIME"
100 PRINT "(02 DWN) ENTER CURRENT HOUR :"
110 INPUT HR$
120 IF VAL(HR$)>23 GOTO 110
130 IF VAL(HR$)>13 GOTO 220
140 PRINT"(DWN) ENTER A FOR A.M."
150 PRINT"(DWN) ENTER P FOR P.M."
160 PRINT"(02 DWN) CHOICE:"
170 GET A$:IF A$=""GOTO 170
180 IF A$="P" THEN GOTO 210
190 IF A$="A" GOTO 220
200 GOTO 170
210 HR$=MID$(STR$(12+VAL(HR$)),2)
220 PRINT "(CLR)(DWN) ENTER CURRENT MINUTES:"
    :INPUT MN$
240 IF VAL(MN$)>59 GOTO 220
250 IF VAL(HR$)<10 THEN HR$="0"+HR$
260 IF VAL(MN$)<10 THEN MN$="0"+MN$
270 T$=HR$+MN$+"00"
280 TIME$=T$
290 PRINT TIME$

295 REM *** ENTER COOKING TIME ***

300 PRINT"(CLR)(02 DWN)"
```

PROGRAM LISTING 10. KITCHEN TIMER

```
310 PRINT"  TOTAL COOKING TIME"
320 INPUT" ENTER HOURS:";HR$
330 INPUT" ENTER MINUTES:";MN$
340 HR=VAL(HR$)
350 MN=VAL(MN$)

355 REM *** ENTER STIR INTERVAL ***

360 PRINT"(CLR)(02 DWN)"
370 PRINT" DO YOU WANT TO BE"
380 PRINT" REMINDED TO STIR"
390 PRINT" BASTE, OR TURN ?"
400 PRINT"(02 DWN)  ENTER Y/N"
410 GET A$:IF A$="" GOTO 410
420 IF A$="Y" GOTO 450
430 IF A$="N" GOTO 500
440 GOTO 410
450 PRINT"(CLR)(02 DWN)"
460 PRINT" ENTER INTERVAL"
470 PRINT" IN MINUTES"
480 INPUT IV$
490 IV=VAL(IV$)
500 PRINT"(CLR)(02 DWN)"
510 PRINT TAB(4)"TIMING CYCLE"
520 PRINT"(02 DWN)"
530 T$=TIME$
540 HN=VAL(LEFT$(T$,2))
550 MP=VAL(MID$(T$,3,2))
560 FH=HN+HR
570 FM=MP+MN
580 FH$=STR$(FH)
590 FM$=STR$(FM)
600 FS$=MID$(T$,5)
610 IF VAL(FH$)<9 THEN FH$="0"+MID$(FH$,2)
620 IF VAL(FM$)<9 THEN FM$="0"+MID$(FM$,2)
630 FT$=FH$+FM$+FS$
640 FT$=MID$(FT$,2)

645 REM *** TIME INTERVAL ***

650 PRINT"(05 DWN) CURRENT TIME: "
660 PRINT"(DWN) FINISH TIME:  ";LEFT$(FT$,4)
670 IF VAL(TIME$)>VAL(FT$) GOTO 820
680 G=VAL(MID$(TIME$,3,2))
690 IF G/IV<>INT(G/IV)THEN POKE 36878,0
    :GOTO710
700 POKE 36878,15
710 POKE CHAR+230,81
720 POKE CSCREEN+230,3
730 FOR N=1 TO 500:NEXT N
740 POKE 36878,0
750 POKE CSCREEN+230,1
760 FOR N=1 TO 500:NEXT N
770 FOR N=1 TO 4
780 POKE CHAR+278+N,ASC(MID$(TIME$,N,1))
```

PROGRAM LISTING 10—CONT. KITCHEN TIMER

```
790 POKE CSCREEN+278+N,6
800 NEXT N
810 GOTO 670

815 REM *** TIMING OVER ***

820 POKE 36879,9
830 POKE 36878,15
840 POKE 36876,255
850 FOR N=1 TO 10
860 POKE 36878,0
870 POKE 36879,24
880 IF PEEK(197)<>64 GOTO 900
890 GOTO 820
900 POKE 36879,122
910 GOTO 910

#
```

PROGRAM LISTING 10—CONT. KITCHEN TIMER

CHAPTER 11

BINGO

Type: Game, unlimited players
Size: 1000 bytes, for any size VIC 20

Here's a short one to give you a breather. After tackling the difficult concept of moving screens (no, this ISN'T basketball!) in the last chapter, you deserve a break.

Does your family fight over who gets to call the bingo numbers when you get together for a friendly game? Is your group or organization looking for a way to add a little space-age appeal to an old-fashioned game?

Here's BINGO, which uses the pseudorandomness of your micro to draw numbers and display them on the screen. All you need are some bingo cards and a few willing participants. Those wanting to use the program before larger groups may want to connect their VIC 20 to a large color television (or a black-and-white set in a pinch). A projection television might be ideal, if the screen could be seen by all in the room easily.

BINGO will choose regulation bingo letter and number combinations and display them on the screen in rows and columns. The letters B-I-N-G-O appear at the top of the screen, and up to 15 characters appear underneath them in the five columns.

The Bingo letter/number combinations are stored in a string array, G$(n), which is defined in line 130 to have 75 elements, one of each of the BINGO numbers. A second array, PO(n), is also defined to keep track of the screen memory positions each number should fill. A routine at lines 280–400 calculates the positions

as each number is turned into its string equivalent and is deposited in the array G$(n) in line 300.

Drawing a number takes place at line 410, where a random number between 1 and 75 is chosen. That element of the array G$(n) becomes the number chosen and the contents are nulled. If that number has already been drawn, the program goes back to line 410 to choose again.

If not, the position for the new entry is chosen by looking at PO(DRAW). Then, the number is POKEd to the screen. Since what we are POKing is not an actual number, but a string representation of that number, it is not possible if, say, B-15 is drawn, to POKE 15 in the proper location under the B. Instead, the CHR$ code for the 1 has to be POKED at position PO, and the CHR$ code for 5 poked at PO+1.

A FOR . . . NEXT loop at 440 takes care of this. The loop repeats from 1 to the length (LEN) of the number drawn. It will make one pass for single digit numbers, and two for two digit numbers. Each time through, PO+N will be POKEd with G+128. G will equal the CHR$ code for the number, and 128 is added in order to print a reverse representation of that number. That is, CHR$(65+128)=reverse A, and so forth.

The correct location in color memory is also calculated (lines 470 and 480), and the color of that location changed from the background color to blue. So, the number chosen is displayed in blue on the screen where all can see it easily.

After the number is printed, a GET A$ loop at lines 500–510 repeats until a key is pressed. At that time, a routine at lines 520–560 POKEs once again at the color memory of the last number chosen, this time placing a 2 there, and changing the color of the number to red. Control then goes to 410 to draw a new number. The effect is to change the last number drawn to red just before the next number is displayed. Bingo players watch the board change as only the current number is displayed in blue.

As mentioned, CSCREEN and CHAR are used for the POKES, to make this program usable on any memory configuration VIC 20. The relevant memory displacement calculation lines, 160 and 170, are included to take care of that chore.

```
10 REM ***********
20 REM *         *
30 REM *  BINGO  *
40 REM *         *
50 REM ***********
60 PRINT "(CLR)"
70 PRINT"(04 DWN)"
80 PRINTTAB(6)"(RVS)BINGO(OFF)(02 DWN)"
90 PRINTTAB(4)"HIT ANY KEY"
100 PRINTTAB(2)"TO DRAW NUMBERS"
110 GET A$:IF A$="" GOTO 110
120 PRINT "(CLR)"
130 DIM G$(75),PO(75)
140 DATA 130,137,142,135,143
150 POKE 36879,174
160 CSCREEN=37888+4*(PEEK(36866)AND128)
170 CHAR=4*(PEEK(36866)AND128)+64
    *(PEEK(36869)AND120)
180 PS=CHAR+2
190 N4=1
200 FOR N=1 TO 5
210 READ A
220 POKE PS,A
230 T2=PS-CHAR
240 POKE CSCREEN+T2,0
250 PS=PS+4
260 NEXT N
270 N3=1

275 REM *** READ NUMBERS TO ARRAY ***

280 FOR N=1 TO 16
290 FOR N2=0 TO 4
300 G$(N3)=STR$(N+(N2*15))
310 N3=N3+1
320 NEXT N2
330 NEXT N
340 FOR N=CHAR+40 TO CHAR+415 STEP 22
350 T=T+1
360 FOR N2=1 TO 5
370 PO(N4)=N+N2*4
380 N4=N4+1
390 NEXT N2
400 NEXT N

405 REM *** DRAW NUMBER ***

410 DRAW=INT(RND(1)*75)+1
420 IF G$(DRAW)="" GOTO 410
430 PO=PO(DRAW)

435 REM *** POKE TO SCREEN ***

440 FOR N=2 TO LEN(G$(DRAW))
450 G=ASC(MID$(G$(DRAW),N,1))
```

PROGRAM LISTING 11. BINGO

```
460 POKE PO+N,G+128
470 T2=PO-CHAR
480 POKE CSCREEN+T2+N,6
490 NEXT N
500 GET A$
510 IF A$="" GOTO 500
520 FOR N=2 TO LEN(G$(DRAW))
530 G=ASC(MID$(G$(DRAW),N,1))
540 POKE PO+N,G
550 POKE PO+N+(CSCREEN-CHAR),2
560 NEXT N
570 G$(DRAW)=""
580 GOTO 410
```

PROGRAM LISTING 11—CONT. BINGO

CHAPTER 12

FLOOR PLANNER

Type: Personal Application
Size: 1000 bytes, for any size VIC 20

Ah, this is a misleading one. FLOOR PLANNER is one of the shortest programs in this book, but it is also one of the most interesting. It introduces the use of the joysticks — the primary tool of all game programmers. With the basics, you can implement your own arcade classics. The VIC 20 has an advantage over some other computers in that its graphics work quickly enough that some solid games can be written even in relatively "slow" BASIC.

FLOOR PLANNER is that implementation of computer assisted drafting that we warned you about in Chapter 6. With it you can draw "walls" on the screen of your VIC 20, finish floor plans, and place "furniture" to see how the layout looks from an aerial view.

In simplest terms, you will be moving a cursor around the screen from box 1 in the VIC memory map (the upper left), to box 484, in the lower right. The starting point, B1, is initially defined as CHAR or, as we have learned, the movable starting point of the VIC character memory. We have to set an upper limit, so the end, E, is defined as CHAR+484. As long as the cursor does not try to move to a lower location than CHAR, or to a higher one than E, we will let it move.

This is done by watching the joysticks. If the joystick is pressed to the right, then the position of the cursor, B1, is incremented

by one. If the joystick is pressed to the left, then the cursor position is decremented by one. An upward movement results in the subtraction of a whole row, 22 positions, from B1, while downward movement is accomplished by adding 22 to B1. That is simple enough.

Each time the program checks the joysticks, the value of B1 is changed to account for the direction in which the joystick is pressed. Then, the program (line 220) POKEs B1 with CURSR, which is the CHR$ code for whatever we want the cursor to be. In most cases, the cursor will be a "+." This allows it to mark off walls with built-in footage markers (each cross-line indicating one foot).

Of course, the cursor will be invisible unless we change color memory as well. So, in line 230, the program pokes B1 + DF (DF being the difference between character memory and color memory) with CL, the color desired.

Unless the "fire" button on either of the joysticks is pressed, this color will be red. If the button is pressed, CL will equal 5, and the color will be cyan; and the cursor will leave an invisible trail behind it. In effect, holding down the fire button leaves blanks, while leaving it untouched causes the cursor to write on the screen.

The cursor character can be changed back and forth from the "+," which signifies walls, to a checkerboard block used to indicate furniture, by pressing any key. FLOOR PLANNER constantly checks to see if a key is depressed by PEEKing location 197. If a 64 is found there, no key was depressed. If a key is struck, some other value will result. At this point, the program drops down to 260–270, where the cursor character is swapped for the one that it was NOT prior to the key depression. So far, we have told you everything except HOW the joystick's position is determined. The VIC 20 joystick is just a box with a number of switches inside. One switch is beneath the red fire button. Four others are at north, south, east, and west positions, and are closed when the joystick is moved in those directions. If some intermediate position is pressed, such as southwest, TWO switches are closed. (Guess which two?)

Because this is a BASIC PROGRAMMING book, I have tried to avoid, wherever possible, discussions of the VIC 20 hardware. If you want to learn about the 6502 chip and other aspects of the VIC's architecture, I recommend the *VIC 20 Programmer's Refer-*

ence Guide (Cat. No. 21948), which is distributed by Howard W. Sams & Co., Inc.

However, to learn how to program the joysticks, it is necessary to understand something of how the VIC 20 interfaces with the real world. Input and output are controlled by a special chip (called the 6522 Versatile Interface Adapter). The only thing you need to know is that this chip has two "ports," which might be thought of as the VIC 20's windows to the outside, each with eight windowpanes.

Like any window, information can pass two ways but, oddly enough, each pane can be used for only one purpose at a time. You can look out through one pane of window A and be seen through another, but not through both simultaneously. You advanced programmers will be interested to know that the eight panes of Port A and Port B are the eight bits of each port. Each of the five switches on the VIC 20 joystick is read through a different bit. The fire button and three of the directional switches are read through register A of VIA chip number one. The last switch can be read through register B of VIA chip number two. What this means is that we need to PEEK two different locations to find out the condition of all five switches. To make things interesting, we must "set" each windowpane for the direction we want information to flow. Two data direction registers (DDR), located at 37139 and 37154, keep track of this direction. By POKing 37139,0, and 37154,127, the registers A and B are set to allow incoming data.

We can then PEEK in two memory locations, 37137 for register A, and 37152 for register B, and obtain the information on the status of the five switches.

Unfortunately, we cannot simply leave the B register set for input from the joystick. If that is done, then the keyboard is dead, and other nasty things can happen. So, for that register, we must set it for input each time we read the joysticks and then restore it with POKE 37154,255 afterwards.

A simple BASIC routine for capturing the joystick information might look something like this:

```
10  POKE 37139,0 : REM Sets Port A for input
20  POKE 37154,127 : REM Sets Port B for input
30  AC = PEEK(37137) : REM Check value at Port A
50  BC = PEEK(37152) : REM Check value at Port B
60  POKE 37139,255 : REM Restore Port B
```

Once we have discovered the values of the byte stored at 37137, and 37152, we are not completely finished. It is, after all, the condition of several BITS within those bytes that we are really interested in. Bits 2,3,4, and 5 from Port A tell us the state of joystick switches 0,1, and 2, and the fire button. Bit 7 from Port B tells us the condition of joystick switch 3.

Fortunately, there is a simple way of checking only one bit within a byte. This is known as Boolean logic and involves performing an AND function with the byte, using a number that, when compared with a byte, will tell whether a desired bit is 1 or 0.

You don't have to understand Boolean logic. Just know that ANDing Port B's value with 128 will tell us about bit 7, and ANDing Port A's value with 4,8,16, and 32 will tell us about the other four switches.

FLOOR PLANNER uses a common routine that is also used in other joystick games in this book. Instead of using the decimal addresses for PEEKing and POKing, we use constants, which are faster for the VIC 20 to handle. The Port B data direction register, which must be repeatedly changed in direction, is defined as DD in line 130. The PEEK addresses for Ports A and B are defined as PA and PB in the same line. And the POKE to 37139, which only must be done once, is entered at that point.

Anytime during the program when we want to know how the joysticks are doing, we summon the routine through a GOSUB 160. In line 160, the Port B DDR is set for input, and the condition of switch no. 3 is read. The AND function is carried out in the same line, and 255 POKed to the DDR to restore the keyboard functions.

Then, a variable P is given the value found in PB, and four AND functions are carried out to determine the status of the fire button (FR) and the other three switches. This can be done very quickly — many times a second — to make the game respond very quickly.

The information we get is in the forms of −1 and +1 values. IF S0,S1,S2,S3 (the up, down, left, and right switches) and FR have the value of zero, then that switch is not closed.

Type in lines 130–210, and then add the following lines:

```
220  GOSUB 160
230  PRINT FR;S0;S1;S2;S3
240  GOTO 220
```

RUN this program with joysticks connected and you will see five columns of numbers scroll past on the screen. Move the joystick. Press the fire button. Every time the button is pressed, a 1 will appear in the first column. Let up on the button and zeroes reappear. Move the joystick right, and a 1 will appear in the fifth column. Left, and a −1 will show up. Upward movement produces a −1 in the second column, and a downward stroke will produce a 1 in the third.

Got it? Then, all we have to do is watch for these changes in values. This is taken care of in lines 310–450. As the joystick is moved, the value of B1 is changed, as we have said, to move the cursor up, down, right, or left.

Who said learning to use the joysticks was all that hard?

```
10 REM *************** *
20 REM *               *
30 REM * FLOOR PLANNER *
40 REM *               *
50 REM *************** *
60 CURSR=91
70 PRINT"(CLR)"
80 CSCREEN=37888+4*(PEEK(36866)AND128)
90 CHAR=4*(PEEK(36866)AND128)+64*
   (PEEK(36869)AND120)
100 E=CHAR+484
110 B1=CHAR
120 DF=CSCREEN-CHAR
130 DD=37154:PA=37137:PB=37152:POKE 37139,0
140 GOTO 220

145 REM *** READ JOYSTICKS ***

160 POKEDD,127:S3=-((PEEK(PB)AND128)=0)
    :POKEDD,255
170 P=PEEK(PA):FR=-((PAND32)=0)
180 S0=((PAND4)=0)
190 S2=((PAND16)=0)
200 S1=-((PAND8)=0)
210 RETURN
220 POKE B1,CURSR
230 POKE B1+DF,CL
240 GOSUB 160
250 CT=PEEK(197):IF CT=64 GOTO 280
260 IF CURSR=102 THEN CURSR=91:GOTO 280
270 IF CURSR=91 THEN CURSR=102
280 IF CL=5 THEN POKE B1+DF,1
290 IF FR<>1 THEN CL=2:GOTO 310
300 CL=5
310 IF S3<>1 GOTO 350
320 B1=B1+1
```

PROGRAM LISTING 12. FLOOR PLANNER

```
330 IF B1>E THEN B1=E
340 GOTO 220
350 IF S2<>-1 GOTO 390
360 B1=B1-1
370 IF B1<CHAR THEN B1=CHAR
380 GOTO 220
390 IF S0<>-1 GOTO 430
400 B1=B1-22
410 IF B1<B THEN B1=B1+22
420 GOTO 220
430 IF S1<>1 GOTO 220
440 B1=B1+22
450 IF B1>E THEN B1=B1-22
460 GOTO 220
```

PROGRAM LISTING 12—CONT. FLOOR PLANNER

CHAPTER 13

COOKIE SHOP

Type: Game, one player
Size: 2700 bytes, for any size VIC 20

The object of this game is to operate a Cookie Shop profitably for one 5-day week. Each 100 cookies baked costs the proprietor $7.50 to make. The cookies may be repriced each day at any price the operator wishes. However, the computer randomly selects an optimum price at the beginning of the week. This ranges from 19 cents to 29 cents.

If a day's selling price is lower than the optimum for the week, the stand will quickly sell out its entire day's stock. The owner could have made and sold more cookies — or charged a higher price and made additional profits.

However, if the selling price chosen is higher than the optimum, consumers will not be willing to buy all the cookies available. Some will be left to go stale. As the margin between the optimum and selling price increases, so does the proportion of stale cookies.

The goal of the proprietor is to deduce the optimum selling price early enough in the week to sell the maximum number of cookies at the highest price the market will bear. The owner may choose to let some cookies go stale on purpose, if the higher price will bring in more profits than a sell-out at the optimum.

In the real world, it's a common practice to raise prices with the knowledge that some customers will be lost, but that profits will be higher on fewer sales. This program's summary at the end

of the game, however, tells the player only what he or she would have made if all the cookies produced had been sold at the optimum price. It does not take into account the maximum number of cookies that could have been made or sold — just the number the player selected to bake for the week.

COOKIE SHOP is a simple program, once you get past the graphics embedded in the print statements. These symbols do nothing more than print an image of a cookie shop on the screen, and fill the store window with cookies at appropriate times. The day of the week is stored in a string array, D$(n).

Then a random number, PR, is chosen in line 630 to represent the optimum price for that particular week. The random number ranges between 0 and 9 and, since 19 is added to it, will produce an optimum price of 19 to 28 cents. You may change this portion of the program, if you wish. Changing the 10 in line 630 to a larger number will increase the range of prices that may be chosen each game. Adding to the 19 will raise the base price to a larger minimum. Then, a FOR . . . NEXT loop from 1 to 5 repeats for the five weekdays of the game.

The program checks, in line 650, to see if the balance of money left, BA, is less than one. If so, the player is bankrupt and the game is over. If not, he or she is asked how many cookies to bake that day. Because it costs $7.50 to make each 100 cookies, the program next checks to see that the player has enough cash left to bake that many. If not, the input is refused, and a message tells just how many cookies CAN be made with the dollars remaining. It would have been possible to print that information to the screen, but the learning value is enhanced by having the player attempt to calculate the number. Alternatively, a sharp player will learn to enter a very large number exactly so that the input will be refused and the maximum calculated by the computer.

The player then enters the price selected for today. If today's price (TP) is less than or equal to the optimum price, then the player is deemed to have sold out his/her entire stock.

The day's profits are calculated and displayed along with a hint that perhaps more could have been charged. If the price was too high, then the overprice (OV) is calculated in line 780, by subtracting the optimum price from today's price. Then, the number that went stale this day is determined (line 790), and those that are sold (DT-S) used to figure profits, if any, or losses for that

day. The new balance is figured, and the loop goes back to allow another day's sales.

The faster a player can guess, or arrive at (by trial and error) the optimum price, the better the chances of finishing the week with the maximum number of sales. At a price of 29 cents and a correct guess on the first round, the potential profits amount to hundreds of thousands of dollars. If your players become too sharp, you can change the value of their initial stake by redefining the value of BA in line 630.

```
10 REM ***************
20 REM *               *
30 REM * COOKIE SHOP *
40 REM *               *
50 REM ***************
60 REM
70 DATA MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY
80 FOR N=1 TO 5
90 READ D$(N)
100 NEXT N
110 GOTO 320
120 POKE 36879,P:RETURN

125 REM *** LOGO ***

130 PRINT"(CLR)"
140 GOSUB 1040
150 PRINT"   (RED)(16 CBM-+)"
160 PRINT"   (02 CBM-+)(PUR)(RVS)COOKIE SHOP
    (OFF)(RED)(03 CBM-+)"
170 PRINT"   (16 CBM-B)"
180 PRINT"   (16 CBM-+)"
190 PRINT"   (02 CBM-+)(WHI)(RVS)       (OFF)
    (RED)(02 CBM-+)(SHF-O)(SHF-P)(03 CBM-+)"
200 PRINT"   (02 CBM-+)(WHI)(RVS)       (OFF)
    (RED)(02 CBM-+)(CBM-G)(CBM-M)(03 CBM-+)"
210 PRINT"   (02 CBM-+)(WHI)(RVS)       (OFF)
    (RED)(02 CBM-+)(CBM-G)(CBM-M)(03 CBM-+)"
220 IF H=1 GOTO 270
230 PRINT"   (04 CBM-+)      (CBM-G)(CBM-M)
    (03 CBM-+)"
240 PRINT"   (04 CBM-+)      (CBM-G)(CBM-M)
    (03  CBM-+)"
250 PRINT"   (04 CBM-+)      (CBM-G)(CBM-M)
    (02 CBM-+)(CBM-+)"
260 GOTO 300

265 REM *** PRINT COOKIES ***

270 PRINT"   (02 CBM-+)(WHI)OOOOOOO(RED)
    (02 CBM-+)(CBM-G)(CBM-M)(03 CBM-+)"
```

PROGRAM LISTING 13. COOKIE SHOP

```
280 PRINT"   (02 CBM-+)(YEL)OOOOOOO(RED)
    (02 CBM-+)(CBM-G)(CBM-M)(03 CBM-+)"
290 PRINT"   (02 CBM-+)(GRN)OOOOOOO(RED)
    (02 CBM-+)(CBM-G)(CBM-M)(03 CBM-+)"
300 PRINT"   (RVS)                  "
310 RETURN
320 P=14:GOSUB 120:H=1:GOSUB 130

325 REM *** INSTRUCTIONS ***

330 PRINT "(02 DWN)     (RVS)INSTRUCTIONS?"
    :GOSUB 1020
340 IF A$="N" GOTO 620
350 PRINT"(CLR)"
360 PRINT"    (RVS) COOKIE SHOP (OFF)"
370 PRINT:PRINT" AS OWNER OF YOUR OWN"
380 PRINT" STORE YOU WILL HAVE"
390 PRINT" FIVE DAYS IN WHICH "
400 PRINT" TO SELL YOUR WARES. "
410 PRINT"(DWN) EACH DAY YOU WILL BE"
420 PRINT" ASKED TO SET A PRICE"
430 PRINT" FOR THAT DAY'S SALES"
440 PRINT" IF YOUR PRICE IS TOO"
450 PRINT" HIGH, CONSUMERS WILL"
460 PRINT" NOT BUY ALL YOU HAVE"
470 PRINT" TO SELL. IF YOU SET "
480 PRINT" THE PRICE TOO LOW"
490 PRINT" YOU LOSE PROFITS."
500 PRINT"(DWN)(WHI)      (RVS)HIT ANY KEY
    (RED)":GOSUB 1020
510 PRINT"(CLR)"
520 PRINT" THE OPTIMUM PRICE IS"
530 PRINT" SET BY THE COMPUTER,"
540 PRINT" BUT KEPT SECRET FROM"
550 PRINT" YOU UNTIL THE END."
560 PRINT"(DWN) THE SOONER YOU GUESS"
570 PRINT" IT, THE BETTER  YOU "
580 PRINT" WILL DO. GOOD LUCK! "
590 PRINT"(DWN) YOU START WITH $100":PRINT
    " AND EACH 100 COOKIES"
600 PRINT" COST YOU $7.50"
610 PRINT"(DWN)(WHI)     (RVS)HIT ANY KEY(OFF)
    (RED)":GOSUB 1020

615 REM *** START GAME ***

620 H=0:P=63:GOSUB 120
630 PR=RND(1)*10+19:PR=INT(PR):BA=100
640 FOR N=1 TO 5
650 IF BA<1 THEN 990
660 GOSUB 130
670 PRINT"(DWN) TODAY IS ";D$(N);"."
680 PRINT "(DWN) HOW MANY COOKIES"
690 INPUT " TO BAKE TODAY";DT
700 IF DT*.075<BA GOTO 720
```

PROGRAM LISTING 13—CONT. COOKIE SHOP

```
710 PRINT"(DWN) YOU CAN MAKE";INT(BA/.075)
    :GOTO 680
720 H=1:GOSUB 130:BA=BA-DT*.075
730 INPUT"(DWN) WHAT PRICE TODAY";TP
740 IF TP<=PR THEN SO=DT:GOTO 860
750 IF TP<0 THEN TP=TP*100
760 IF TP<=PR THEN SO=DT:GOTO 860
770 IF TP>200 THEN S=DT:SO=0:GOTO 830
780 OV=TP-PR
790 S=(OV/PR+.2)*DT:S=INT(S)
800 IF S>DT THEN S=DT
810 SO=DT-S
820 IF SO<1 THEN SO=0

825 REM *** PRICE HIGH ***

830 GOSUB 130
840 PRINT"(DWN) PRICE WAS TOO HIGH!"
850 PRINT " YOU SOLD";SO:PRINT S;"WENT STALE"
    :GOTO 890

855 REM *** PRICE LOW ***

860 H=0:GOSUB 130:PRINT" YOU SOLD OUT!"
870 PRINT" PERHAPS YOU SHOULD "
880 PRINT" CHARGE MORE TOMORROW?"
890 TA=SO/100*TP:BA=BA+TA:PRINT "(DWN)
    SALES:      $";INT(TA)
900 PRINT " BALANCE  : $";INT(BA)
910 PRINT "(DWN)     (RVS)HIT ANY KEY(OFF)"
920 GOSUB 1020
930 H=0:NEXT N

935 REM *** PRINT RESULTS ***

940 P=26:GOSUB 120:GOSUB 130
950 PRINT"(DWN)     (RVS)FINAL TOTAL(OFF)"
960 PRINT"(DWN) WEEK'S SALES $";INT(BA)
970 PRINT" OPTIMUM PRICE "
980 PRINT" WAS ";PR;"CENTS.":GOTO 1010
990 P=14:GOSUB 120:GOSUB130
1000 PRINT "(DWN) YOU WENT BANKRUPT!"
1010 PRINT"(DWN)  TRY AGAIN SOON!"
     :GOSUB 1020:RUN
1020 V=8:T=241:GOSUB 1040
1030 GET A$:IF A$="" GOTO 1030

1035 REM *** SOUND ROUTINE ***

1040 POKE 36878,V
1050 POKE 36875,T
1060 FOR U=1 TO 200:NEXT U
1070 POKE 36875,0
1080 RETURN

#
```

PROGRAM LISTING 13—CONT. COOKIE SHOP

CHAPTER 14

BLACK BOOK

Type: Personal Application
Size: 1400 bytes, for any size VIC 20

BLACK BOOK is a simple name and address data file that demonstrates how to keep a data base without using either disk data files, or cassette files. Sometimes we like to keep track of information that is entered into a program permanently. Storing data on cassette or disk with the VIC 20 is relatively simple, but each method does have its drawbacks.

To use disk files, you must have a disk drive. Most models available are rather expensive, about four times what the VIC 20 computer is at discount. Even though approximately one million of the computers have been sold, I don't imagine that nearly one million of the Commodore Datasette recorders have been purchased. And the number of disk drives in use with the VIC 20 is very much lower. So, VIC disk files are far from a universal tool, available to everyone.

Everyone reading this book with ambitions of learning BASIC programming should have a program recorder, however. But, because of the header written on data files, writing them to cassette and loading data from tape is relatively slow. In addition, how do you know your data was recorded successfully? The VIC 20 uses special circuitry that makes its cassette recorder much more reliable than those used on some other computer systems. Because of this, it can use much less expensive tapes than other computers. These tapes do have dropouts, or places with no iron

oxide available to record information. That is why it is wise to VERIFY any program that is SAVEd.

Because it is so simple to check a program, why not simply store data within a program itself? At the end of the RUN, SAVE the new program, VERIFY it, and you can be assured that your information is safely transferred to tape. BLACK BOOK allows entering the name, phone number, and birthday of an acquaintance in the form of DATA lines. It shows you the lines already entered so that the new one may be placed at the end. It also prompts you to make one other change that tells the program how many items of data have been entered.

Then, in running the program, you can search for a given name. The VIC 20 will loop through all the names on file, display the name, birthday, and phone number of the person you wish, or tell you that the name is not in the file.

The program first reads the DATA into a two dimensional string array, DTA$(row,column). Earlier in the book we used single dimensional arrays, which we likened to a row of empty boxes. If you had such a row in your living room, it might stretch clear across the room. Now, picture four such rows side by side on the floor of your living room. You might tell someone to check a box by having them look in row three, column two, the way they might check an automobile mileage chart.

Two dimensional arrays also have rows and columns. Each row consists of information about one person: the name, phone number, or birthday. Each column of the array consists of a specific type of data, such as birthdays. So, Row no. 1, Column no. 1, would have the name of the first person in your data base. Row no. 4, Column no. 3, would have the birthday of the fourth person in the list.

Two nested FOR . . . NEXT loops read all this data into the proper places in the array DTA$(row, column). In line 1020, the first loop starts from 1 to R, which is the number of rows, and which is defined in line 1000. Then, the second loop starts from 1 to three, the number of columns. Each time through the second loop one of the data items is stored in DTA$(ROW,COLUMN).

When all three have been read for a particular person, the program advances to line 1060, which sends control back to line 1020 for the next ROW. After all the data has been read, a menu is presented, which offers the choice of adding a listing or retrieving an existing name. If the latter is checked, control goes to

line 1240 and the user is asked to enter a name to search for. This string is stored in variable NME$, and a FOR . . . NEXT loop from 1 to R looks at column no. 1 of each row until a match is found.

If it is, the program goes to line 1350, where the data is displayed on separate lines. If no match is found, the computer tells us the bad news. In either case, the program returns to the main menu. If we want to add a name, instructions appear, telling the user to add the DATA line to the last one shown and to change line 1000 so that R is defined as one larger. The new value of R (R + 1) is displayed — to make this very simple. Then, the program lists all the lines from 10 to 999 and turns the computer over to the user in command mode.

The program line number chosen should be one larger than the last number used (i.e., on the initial RUN, 11 should be used). This will make it possible to fit the maximum number of DATA lines to be entered. Actually, the DATA can be inserted anywhere in the program, but it is much neater if they are located all in one consecutive block.

```
1 REM ***************
2 REM *               *
3 REM *  BLACK BOOK   *
4 REM *               *
5 REM ***************
6 GOTO 1000

7 REM *** INSERT DATA LINES HERE ***

10 DATA NAME,PHONE,BIRTHDAY
1000 R=1
1010 DIM DTA$(R,3):POKE 36879,120

1015 REM *** READ DATA ***

1020 FOR ROW=1 TO R
1030 FOR COLUMN=1 TO 3
1040 READ DTA$(ROW,COLUMN)
1050 NEXT COLUMN
1060 NEXT ROW

1065 REM *** MENU ***

1070 PRINT "(CLR)(02 DWN)"
1080 PRINT "  LITTLE (RVS)(BLK)BLACK(OFF)
     (BLU) BOOK"
1090 PRINT "(02 DWN)  1. ADD LISTING"
1100 PRINT "(02 DWN)  2. RETRIEVE"
1110 GET A$:IF A$="" GOTO 1110
```

PROGRAM LISTING 14. BLACK BOOK

```
1120 A=VAL(A$)
1130 IF A<1 OR A>2 GOTO 1110
1140 ON A GOTO 1150,1240

1145 REM *** ADD LISTING ***

1150 PRINT "(CLR)(DWN)"
1160 PRINT " TO ADD LISTING ENTER"
1170 PRINT " DATA LINE FOLLOWING"
1180 PRINT " LAST ONE SHOWN."
1190 PRINT "(02 DWN)   HIT ANY KEY"
1200 GET A$:IF A$="" GOTO 1200
1210 PRINT"(CLR)(DWN)TYPE THIS LINE FIRST:"
1220 PRINT" 1000 R=";R+1
1230 LIST 10-999

1235 REM *** RETRIEVE ***

1240 PRINT"(CLR)(02 DWN)"
1250 PRINT" ENTER NAME TO"
1260 PRINT" SEARCH FOR:(02 DWN)"
1270 INPUT NME$
1280 FOR ROW=1 TO R
1290 IF DTA$(ROW,1)=NME$ THEN GOTO 1350
1300 NEXT ROW
1310 PRINT"(CLR)(02 DWN)   == (RVS)
     NOT FOUND(OFF) =="
1320 PRINT"(02 DWN)    HIT ANY KEY"
1330 GET A$:IF A$="" GOTO 1330
1340 GOTO 1070
1350 PRINT"(CLR)(02 DWN) (RVS)(RED)
     FOUND:(02 DWN)"
1360 PRINT" NAME:";DTA$(ROW,1)
1370 PRINT"(DWN) PHONE:";DTA$(ROW,2)
1380 PRINT"(DWN) BIRTH:";DTA$(ROW,3)
1390 PRINT"(02 DWN)(BLU)      HIT ANY KEY"
1400 GET A$:IF A$="" GOTO 1400
1410 GOTO 1070

#
```

PROGRAM LISTING 14—CONT. BLACK BOOK

CHAPTER 15

MOTOR RACE

Type: Game, one player
Size: 2400 bytes, for unexpanded VIC 20 only

MOTOR RACE is a relatively simple game. The player steers a racing car back and forth laterally, attempting to avoid pylons at the side of the road and oncoming cars which he or she overtakes. The width of the road may be set, as well as the speed, the number of other cars, and the number of crashes allowed. Realistic motor sounds emanate from the screen, and some relatively tricky driving is possible. Sneaky drivers can attempt to maneuver past the pylons off the race course entirely but, since there are no obstacles to cause problems, this is considered unsportsmanlike. This game is for the UNEXPANDED VIC 20 computers only, because it makes use of a special programming trick.

If you will review your instructions for the VIC, you will notice that two types of racing cars and the pylons are NOT provided among the special graphics characters. To achieve them, we had to alter the characters in the VIC's character set. This is one of the most flexible tricks you can use, because it is possible to construct flying saucers, monsters, foreign language symbols, or any character you wish.

First, let's look at how the VIC 20 thinks of its existing characters. These are constructed on an eight by eight-dot matrix, with the first and last columns usually empty, and the bottom row empty. That arrangement leaves space between each character and the next. Look at how a letter "A" is put to-

gether and you will see what I mean. Each "0" is considered a blank space and each "1" a dot that is filled in:

```
0 0 0 1 1 0 0 0
0 0 1 0 0 1 0 0
0 1 0 0 0 0 1 0
0 1 1 1 1 1 1 0
0 1 0 0 0 0 1 0
0 1 0 0 0 0 1 0
0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0
```

See the letter "A" in that pattern? Since each row is eight characters across, and each character is either a zero or a one, it is convenient to think of each row as a byte, and to store it that way in memory. Eight consecutive bytes will store the eight rows needed to describe a given character.

This is exactly what the VIC 20 does. It so happens that the information on the letter "A" begins at memory location 32776, and continues for eight bytes. The first line in our dot matrix pattern for "A", 00011000 in binary, is 24 in decimal. If you type PEEK(32776), the VIC 20 will return the value 24. Similar 8-byte groups are found in memory to tell the computer how to form all the alpha and graphics characters, including reversed characters.

Wouldn't it be simple to just POKE new values into memory to change characters to new patterns? For example, since the letter "B" is stored at 32784 to 32791, by typing:

```
10 FOR N=32784 to 32791
20 POKE N,126
30 NEXT N
```

We could turn the letter "B" into a block (since 126 decimal equals 01111110 binary). Unfortunately, things are not that simple. Characters are actually stored in ROM. We can READ the information, but not change it.

However, when the VIC 20 wants to find out how to build a given character, it does not go directly to the proper ROM location. Instead, it checks 36879, which tells it where to find the beginning of the character memory. Normally, a value of 240 or 242 is there, which points to the locations in ROM that store the descriptions of upper-case and graphics characters, or upper-

/lower-case characters, respectively. Type POKE 36879,242 when in upper-case/graphics mode, to see what happens.

You might have guessed that 36879 is a RAM location. Because it is, the programmer can POKE in something other than 240 or 242, to send the VIC 20 off to some OTHER area of RAM to find its character information. If you do that, you must arrange to have ALL the characters you want to use stored in the proper form. The VIC 20 will not go back and forth, looking in ROM for some characters, and RAM for others. Normally, this is accomplished by COPYING from ROM the information about all the characters you want to use, and then modifying only those you want changed.

That is what is done in MOTOR RACE. The first step is to POKE 36869 with 255 (line 60). This tells the VIC 20 to look not at the normal location for its character information, but to start at 7168 decimal instead. Because this location is BASIC RAM, we have to protect it by lowering the top of RAM memory. That is accomplished by POKing 28 into 52 and 56 decimal, which are the registers that keep track of how much RAM is available for programs. Once those pointers have been changed, your program will not have any locations above 7167 available. The new character set will be safe.

Next, we will copy 64 characters from ROM into the protected RAM locations. This is done in line 70, which is a FOR . . . NEXT loop from 7168 to 7175. Each time through the loop the program PEEKs in the ROM, extracts a byte, and POKEs it in the next location of the protected area.

If the program did nothing more than that, then the character set would look exactly the same, except that the VIC 20 would be obtaining the information from a different place. Instead, we will POKE some new data into the locations for some selected characters that are not needed by MOTOR RACE. These characters are "at" sign (@), the exclamation point (!), and the greater than symbol (>). The characters chosen now are defined beginning at 7168, 7432, and 7664, respectively.

We must POKE new values into each of the eight bytes defining those characters. The values are determined simply by laying out an 8 by 8 grid, and changing the binary values obtained to the decimal equivalents.

For example, a racing car might be designed like this:

Binary	Decimal
0 0 1 1 1 1 0 0	60
1 1 1 1 1 1 1 1	255
1 1 1 1 1 1 1 1	255
0 0 1 1 1 1 0 0	60
0 0 1 1 1 1 0 0	60
1 1 1 1 1 1 1 1	255
1 1 1 1 1 1 1 1	255
0 0 0 1 1 0 0 0	24
0 0 1 1 1 1 0 0	60

By entering those eight numbers in a DATA line (line 200) and then POKing them into eight memory locations beginning at 7168, every time we print an @ to the screen an image of the race car will appear instead. That is exactly what is done, thus producing two different types of cars, and one type of pylon.

Most of the other aspects of this game have been covered in previous chapters. POKEs to sound registers produce the motor sound and the sound of crashes. TI$ is set to zero at the start of a race (line 190) and checked at the end to see how long the race lasted.

The user can set various parameters, such as speed (SP), which is nothing more than a variable used to provide a delay through a FOR . . . NEXT loop between movements of the cars on the screen. This delay occurs in line 410. The variable W sets the width of the road. Line 220, for example, prints pylons (indicated in the program listings as ''>'') by printing one pylon, then TABing 8, plus the width stored in W. The variable is also used to position the oncoming cars.

Their frequency (CA) is set by user, but the placement and their timing are determined by random numbers selected by the computer. Crashes are created by appropriate noises and flashing of the player's car.

```
10 REM **************
20 REM *              *
30 REM * MOTOR RACE *
40 REM *              *
50 REM **************

55 REM *** CHANGE CHARACTERS ***
```

PROGRAM LISTING 15. MOTOR RACE

```
60 PRINT"(CLR)":GOSUB530:POKE 36869,255
   :POKE52,28:POKE56,28
70 FOR I=7168TO7679:POKEI,PEEK(I+25600)
   :NEXTI
80 FORN=7168TO7175:READ H:POKEN,H:NEXTN
90 FORN=7432TO 7439:READ H:POKEN,H:NEXTN
100 FOR N=7664 TO 7671:READH:POKEN,H:NEXTN
110 VOICE=36874:SOUND=255:V=10:BRDR=BRDR
    :VLUME=36878
120 POKE BRDR,24
130 DD=37154:PA=37137:PB=37152:PO=3
    :FG=1:U1=7680:U2=7702:U3=8164
140 POKE BRDR,14
150 POKE 37139,0
160 POKE VLUME,V
170 POKE VOICE,SOUND
180 T=1:A=4
190 TI$="000000"
200 DATA 60,255,255,60,60,255,255,24,
    60,255,255,60,255,24,60,255
210 DATA 0,24,60,60,126,126,255,255

215 REM *** PRINT PYLONS ***

220 FORN=1TO23:PRINTTAB(8)">";TAB(8+W)">"
    :NEXTN
230 T=8:PO=10:GOTO 280

235 REM *** POSITION ENEMY CARS ***

240 D=INT(RND(1)*2)+1
250 J=INT(RND(1)*CA)+1:IF J<>2THEN
    GOTO 300
260 FA=INT(RND(1)*3)+1:FA=FA*D
270 F4=T+INT(W/2)+FA:IF F4>T+W
    OR F4<T GOTO 300
280 IF F4=PR GOTO300
290 POKEU3+F4,0
300 PR=F4:IFD=2THEND=-1
310 IFT+W+D>18GOTO340
320 IF T+D<1GOTO 340
330 T=T+D
340 GOSUB 400
350 POKE VOICE,0
360 PRINTTAB(T)">";TAB(T+W)">";
370 PRINT" "
380 POKE VOICE,SOUND
390 GOTO 240
400 GOSUB 870

405 REM *** DELAY ACCORDING TO SPEED ***

410 FOR O=1 TO SP:NEXTO
420 POKEU1+PO,32
```

PROGRAM LISTING 15—CONT. MOTOR RACE

```
425 REM *** FIND DIRECTION TO GO ***

430 IF S2=-1 THEN PO=PO-1
440 IF S3=-1 THEN PO=PO+1
450 IF PO<1 THEN PO=1
460 IF PO>21 THEN PO=21
470 LF=PEEK(U2+PO):IFLF=32 GOTO 500
480 IFLF=33GOTO500
490 GOSUB900
500 POKEU2+PO,33
510 POKEU2+PO,33
520 RETURN

525 REM *** LOGO ***

530 PRINT"(CLR)(02 DWN)"
540 PRINT"      (BLU)(RVS)MOTOR RACE
    (OFF)(02 DWN)"
550 PRINT"(BLK)(22 CBM-B)";
560 PRINT"(RVS)(GRN)                              ";
570 PRINT"(RED)COPYRIGHT 1982(RVS)(GRN)    ";
580 PRINT"    (RED)(RVS)DAVID D. BUSCH
    (GRN)                          (OFF)";
590 PRINT"(BLK)(22 CBM-B)
600 PRINT"(02 DWN)  (PUR)USE ATARI JOYSTICKS
    (02 DWN)"
610 PRINT"(02 DWN)     (PUR) (RVS)HIT ANY
    KEY(RVS)"
620 GET A$:IF A$=""GOTO 620

625 REM *** SET PARAMETERS ***

630 PRINT"(CLR)(02 DWN)(BLU)"
640 PRINT"  WHAT SPEED?(DWN)"
650 PRINT" 1 (FAST) TO 9(SLOW)"
660 GET A$:IF A$=""GOTO 660
670 SP=VAL(A$)
680 IF SP<1 GOTO660
690 SP=SP*2
700 PRINT"(CLR)(02 DWN) HOW WIDE WOULD
    (DWN)"
710 PRINT" YOU LIKE THE ROAD?(DWN)"
720 PRINT"1 (NARROW) TO 9 (WIDE)"
730 GET A$:IF A$=""GOTO 730
740 W=VAL(A$)
750 IF W<1 OR W>9 GOTO 730
760 PRINT"(CLR)(02 DWN) HOW MANY CRASHES"
770 PRINT"(DWN) BEFORE  YOUR"
780 PRINT"(DWN) GAME IS OVER?(DWN)"
790 INPUT MC$:MC=VAL(MC$):IF MC<1
    OR MC>50 GOTO 770
800 PRINT"(CLR)(02 DWN) OTHER CARS
    ON ROAD?(DWN)"
810 PRINT"1 (MANY) TO 20 (FEW)(02 DWN)"
820 INPUT CARS$
```

PROGRAM LISTING 15—CONT. MOTOR RACE

```
830 CAR=VAL(CAR$)
840 IF CAR<1OR CAR>20GOTO 800
850 CAR=CAR+1
860 PRINT"(CLR)":RETURN

865 REM *** READ JOYSTICKS ***

870 POKEDD,127:S3=((PEEK(PB)ANDl28)=0)
    :POKEDD,255
880 P=PEEK(PA):S2=((PANDl6)=0)
890 RETURN

895 REM *** CRASH!! ***

900 POKE VOICE,0
910 POKE 38422+PO+Xl,2:POKEU2+PO+Xl,33
920 POKE VLUME,15
930 POKE36877,255
940 FORF=1 TO 100
950 POKE BRDR,104:POKE BRDR,14:NEXTF
960 CR=CR+1
970 POKE36877,0
980 POKE VLUME,8
990 IF CR=MCTHENGOTOl040
1000 POKE VOICE,SOUND
1010 POKE VLUME,V
1020 POKE 38422+PO+Xl,1
1030 RETURN

1034 REM *** RACE OVER ***

1040 FT$=TI$
1050 PRINT"(CLR)(02 DWN)    RACE OVER.(DWN)"
1060 PRINT" YOU HAVE CRASHED(DWN)"
1070 PRINT CR;" TIMES.(DWN)"
1080 PRINT"(DWN) YOUR TIME:(DWN)"
1090 FT$=MID$(FT$,3,2)+":"+MID$(FT$,5)
1100 PRINT TAB(5)FT$
1110 PRINT"(02 DWN) TRY AGAIN?"
1120 GET A$:IF A$=""GOTO 1120
1130 IF A$="Y"THEN RUN

#
```

PROGRAM LISTING 15—CONT. MOTOR RACE

CHAPTER 16

VIC ORGAN

Type: Personal Application
Size: 2000 bytes, for unexpanded VIC 20s only

Sound has been used extensively in many programs in this book. Using the VIC 20 as an organ or electric piano requires only that we not select sounds at random but, rather, assign them in an order roughly approximating the musical scale.

Fortunately, the VIC 20's three voices will come close to actual notes simply by POKing certain numbers into the three registers. These numbers are stored in DATA lines (beginning at line 230) and READ into an array, P(n). The letter names of the keys are stored in another array, N(n), while a third, CL(n), is used to keep track of the screen locations of the video representations of the keys. A symbol is flashed on the screen through a POKE whenever a given key is pressed, its location being determined from CL(n).

VIC ORGAN scans the keyboard and, when certain keys are pressed, POKEs those notes for us. The "home" row of the keyboard (ASDFGHJKL:;=) is assigned to the natural notes, while appropriate letters on the row above are set for sharps and flats.

Pressing any of those keys will produce the correct tone. Touching one of the top three special function keys will change the tone from highest voice, to medium, and lowest. Hitting the fourth function key will cause the computer to play the song stored in memory, so far.

Touching one of the number keys will lengthen or shorten the timing of ALL the notes in the song. Striking the space bar will enter a rest (pause) of the same length. The CLR key is used to clear the song in memory to start over. Otherwise, after playing a song and typing new notes, the added notes will be appended onto the old melody.

The registers which will be POKEd are given variable names that make them simpler to comprehend when viewing the listing. Instead of POKEing a value from 0 to 15 into 36878, that number is called VOLUME, and we POKE VOLUME, 15, for example. The same is done for the voice registers, which are called, LOW, MEDIUM, and HIGH, respectively.

Each key is assigned a value which, when POKed into one of the voice registers, produces an appropriate note. As these keys are pressed by the player, the POKE is made, producing the note and, in addition, the value is stored in an array, NT(n), which can hold a song up to 200 notes long. When the F7 key is pressed and detected, in line 136, the program goes to line 560, where the song is played. Voice is changed between lines 510–540, and the actual length of the note played altered by a delay loop of from 1 to LTH, with LTH being the length (lines 580–590).

```
10 REM *************
20 REM *           *
30 REM * VIC ORGAN *
40 REM *           *
50 REM *************
60 DIM P(22),N(22),NT(200),CL(19)
70 VOLUME=36878
80 LOW=36874
90 MEDIUM=36875
100 HIGH=36876
110 VCE=LOW:LTH=200
120 SET=15

125 REM *** READ NOTES INTO ARRAY ***

130 FOR N= 1 TO 22
140 READ P(N)
150 NEXT N

155 REM *** READ KEY NAMES ***

160 FOR N=1 TO 22
170 READ N(N)
180 NEXT N
```

PROGRAM LISTING 16. VIC ORGAN

```
185 REM *** READ SCREEN POSITIONS ***

190 FOR N=1 TO 18
200 READ CL(N)
210 NEXT N
220 POKE 36879,105:GOTO740
230 DATA 135,143,147,151,159,163,167,175,179
    183,187,191,195,199,201,203,207,209,
240 DATA 212,215,217,219,65,87,83,69,68,
    70,84,71,89,72,85,
250 DATA 74,75,79,76,80,58,59,42,61,94,13
260 DATA 8143,8056,8145,8058,8147,8149,8062
270 DATA 8151,8064,8153,8066,8155,8157,8070,
    8159,8072,8161,8163

275 REM *** WAIT FOR INPUT ***

280 GET A$:IF A$=""GOTO280
290 A=ASC(A$)
300 IF A=19 THEN NT=0
310 IF A>132 THEN GOSUB 510
320 IF A>48 AND A<58 THEN LTH=(A-48)*100
330 IF A=32 THEN NT=NT+1:NT(NT)=0:GOTO 280
340 FOR N=1 TO 22
350 IF A$=CHR$(N(N))GOTO 380
360 NEXT N
370 GOTO 280
380 POKE VOLUME,SET
390 IF N<19 THEN POKE CL(N),42
400 POKE VCE,P(N)
410 FOR L=1 TO LTH
420 NEXT L
430 POKE VCE,0
440 NT=NT+1
450 IF NT=199 THEN GOSUB 560:NT=0
460 NT(NT)=P(N)
470 IF N>18 THEN GOTO 280
480 POKE CL(N),32
490 N=0
500 GOTO 280

505 REM  *** RESPOND TO FUNCTION KEYS ***

510 IF A=133 THEN VCE=LOW:RETURN
520 IF A=134 THEN VCE=MEDIUM:RETURN
530 IF A=135 THEN VCE=HIGH:RETURN
540 IF A=136 THEN GOTO 560
550 RETURN

555 REM *** PLAY SONG SO FAR ***

560 FOR E=1 TO NT
570 POKE VCE,NT(E)
580 FOR G=1 TO LTH
590 NEXT G
```

PROGRAM LISTING 16—CONT. VIC ORGAN

```
600 POKE VCE,0
610 NEXT E
620 RETURN

625 REM *** PRINT INSTRUCTIONS ***

630 POKE 36879,105:PRINT"(CLR)(DWN)"
640 PRINT"    (RVS)(YEL)<CLR>(OFF) = START
    OVER(DWN)"
650 PRINT" (RVS)(YEL)1(OFF) (SHORT)TO (RVS)
    9(OFF) (LONG)(DWN)"
660 PRINT"     (RVS)<SPACE>(OFF) = REST(DWN)"
670 PRINT"     (RVS)F1(OFF)  LOW VOICE"
680 PRINT"     (RVS)F3(OFF)  MEDIUM VOICE"
690 PRINT"     (RVS)F5(OFF)  HIGH VOICE"
700 PRINT"(02 DWN)     (RVS)F7(OFF)  PLAY SONG
    (DWN)":PRINT"(22 CBM-+(DWN)";
710 PRINT"(DWN)(BLK)  (RVS)W(OFF) (RVS)E
    (OFF)  (OFF) (RVS)T(OFF) (RVS)Y(OFF) (RVS)
    U(OFF)  (OFF) (RVS)O(OFF) (RVS)P(OFF)  (OFF)"
720 PRINT"(DWN)(WHI) (RVS)A(OFF) (RVS)S
    (OFF) (RVS)D(OFF) (RVS)F(OFF) (RVS)G
    (OFF) (RVS)H(OFF) (RVS)J(OFF) (RVS)K
    (OFF) (RVS)L(OFF) (RVS):(OFF) (RVS);(OFF) "
730 GOTO 280

735 REM *** LOGO ***

740 PRINT"(CLR)(02 DWN)"
750 PRINT"(DWN)(BLK)  (RVS) (OFF) (RVS) (OFF
    )  (OFF) (RVS)V(OFF) (RVS)I(OFF) (RVS)C
    (OFF)  (OFF) (RVS) (OFF) (RVS) (OFF)  (OFF)"
760 PRINT"(DWN)(WHI) (RVS) (OFF) (RVS) (OFF) (
    RVS) (OFF) (RVS)O(OFF) (RVS)R(OFF) (RVS)G
    (OFF) (RVS)A(OFF) (RVS)N(OFF) (RVS) (OFF
    ) (RVS) (OFF)  "
770 PRINT"(04 DWN)     (RVS)(YEL)COPYRIGHT
    1983(DWN)"
780 PRINT"     (RVS)DAVID D. BUSCH"
790 PRINT"(03 DWN)(OFF)       (RVS)(GRN)HIT
    ANY KEY"
800 GET A$:IF A$=""GOTO 800
810 PRINT"(CLR)":GOTO 630

#
```

PROGRAM LISTING 16—CONT. VIC ORGAN

CHAPTER 17

BARRIER RUN

Type: Game, one player
Size: 1700 bytes, for all VIC 20s

BARRIER RUN is a joystick game that, unlike MOTOR RACE, lets you move the object on the screen in all four directions. In this respect, it is very similar to FLOOR PLAN. In fact, the principle behind moving the "head" of the player's snake is identical.

The object of the game is to stay alive for as long as possible by moving around the screen, avoiding obstacles, and your own "tail." The computer helps out a bit by randomly demolishing holes on the screen, making gaps that can be used to escape through walls, or the snake's trail. The big difference between this game and FLOOR PLAN is that the objective always is in motion on the screen. If the joystick is not passed in one of four directions, then the head continues in the same direction that it was previously going.

That direction is stored in variable DELTA. After each check of the joystick if a directional switch has been closed, then DELTA is changed to that new direction. If not, then the former value of DELTA is used to determine the new position, B1, which is POKEd both in the character memory and color memory to move the snake one position further.

A check is made to see if B1 is a space. If not, the program goes to the crash routine at line 670, where appropriate flashes and noises are made, the elapsed time determined, and a comparison made to see if a new high score (line 800) has been achieved.

As the snake moves, a random number is chosen in line 350, in the range 0 to 484, and this position in character memory POKEd with a 32, which produces a space. If a space already exists there, the effect is nil. But if a wall or tail segment is located in that position, a hole appears which can be traversed. As the screen fills, fewer spaces are left, and so the effect of the random demolition becomes more pronounced.

To add a little play value, the snake starts off slowly each round, from a random position at the left side of the screen, chosen in line 250. A delay loop, with DLAY beginning at a value of 200, is run between each move, with DLAY being reduced by 6 each time. After about 33 movements, DLAY becomes a minus number, and thus the snake moves at full speed.

```
10 REM ***************
20 REM *               *
30 REM * BARRIER RUN *
40 REM *               *
50 REM ***************
60 PRINT"(CLR)"
70 PRINTTAB(4)"(02 DWN)(RVS)(RED)BARRIER RUN"
80 PRINT"(02 DWN)  USE JOYSTICKS"
90 PRINT"  TO AVOID HITTING "
100 PRINT"  OBSTACLES.  GAIN"
110 PRINT"  POINTS FOR EACH"
120 PRINT"  SECOND YOU STAY"
130 PRINT"  ALIVE!"
140 PRINT"(02 DWN)    (RVS)HIT ANY KEY"
150 GET A$:IF A$=""GOTO 150
160 POKE 36879,90
170 PRINT"(CLR)"
180 CSCREEN=37888+4*(PEEK(36866)AND128)
190 CHAR=4*(PEEK(36866)AND128)+64
    *(PEEK(36869)AND120):B=CHAR:E=CHAR+484
200 DF=CSCREEN-CHAR
210 DD=37154
220 PA=37137
230 PB=37152

235 REM *** START TURN ***

240 DELTA=1
250 F=INT(RND(1)*21)
260 DLAY=200
270 B1=(CHAR+F*22)-1

275 REM *** SET RIGHT SIDE WALL ***

280 FOR N=B+21 TO E+21 STEP 22
290 POKE N,81
```

PROGRAM LISTING 17. BARRIER RUN

```
300 POKE N+DF,0
310 NEXT N
320 BEGN=TI
330 GOSUB 580
340 GOSUB 850

345 REM *** CHOOSE LOCATION TO BLANK ***

350 I=INT(RND(1)*484)
360 POKE CHAR+I,32
370 FOR J=1TODLAY:NEXTJ
380 DLAY=DLAY-6

385 REM *** FIND NEW DIRECTION ***
390 IF S0<>-1 GOTO 420
400 DELTA=-22:IF B1+DELTA<B THEN DELTA=0
410 GOTO 510
420 IF S1<>1 GOTO 450
430 DELTA=22:IF B1+DELTA>E THEN DELTA=0
440 GOTO 510
450 IF S2<>-1 GOTO 480
460 DELTA=-1:IF B1+DELTA<B THEN DELTA=0
470 GOTO 510
480 IF S3<>1 GOTO 510
490 DELTA=1:IF B1+DELTA>E THEN DELTA=0
500 GOTO 510
510 B1=B1+DELTA
520 IF PEEK(B1)<>32 GOTO 670

525 REM *** MOVE DOT ***

530 POKE B1,81
540 POKE B1+DF,7
550 POKE B1+DF-DELTA,2
560 GOTO 330

565 REM *** READ JOYSTICKS ***

580 POKE DD,127
590 S3=-((PEEK(PB)AND 128)=0)
600 POKE DD,255
610 P=PEEK(PA)
620 S1=-((PAND8)=0)
630 S2=((PAND16)=0)
640 S0=((PAND4)=0)
650 FR=-((PAND32)=0)
660 RETURN

665 REM *** CRASH !!! ***

670 FOR N=1 TO 50
680 POKE 36877,200
690 POKE B1+DF-DELTA,7
700 POKE B1+DF-DELTA,1
710 POKE B1+DF-DELTA,2
```

PROGRAM LISTING 17—CONT. BARRIER RUN

```
720 POKE 36877,0
730 NEXT N

735 REM *** PRINT RESULTS ***

740 PRINT"(CLR)"
750 FSH=TI
760 ET=(FSH-BEGN)/60
770 ET=INT(ET*100)/100
780 PRINT" (RVS)(RED)ELAPSED TIME:(02 DWN)"
790 PRINT ET;" (RVS)SECONDS"
800 IF HS<ET THEN PRINT"(02 DWN) NEW
    HIGH SCORE!!":HS=ET
810 FOR N=1 TO 1000
820 NEXT N
830 PRINT"(CLR)"
840 GOTO 240

845 REM *** SOUND ROUTINE ***

850 POKE 36878,15
860 POKE 36874,200
870 POKE 36874,0
880 RETURN

#
```

PROGRAM LISTING 17—CONT. BARRIER RUN

CHAPTER 18

POP !

Type: Game, one player
Size: 2700 bytes, for all VIC 20s

By now, you should be familiar with most of the special features of the VIC 20. Using sound in programs should be a snap; special function keys summoned at the drop of a CHR$ (133–140). We've conquered the evil joysticks, and even tampered with the hallowed inner sanctum of the character memory. We'll finish off the major programs of this book with POP !, which is a joystick extravaganza that uses most of the VIC 20's capabilities.

The player's missile firing base is installed along the bottom of the screen. It may be moved from side to side by means of the joystick. Pressing the FIRE button unleashes a deadly arrow, which ascends inexorably skyward. Only a single arrow may be on the screen at one time, but they can be steered, using the joystick, and wraparound from one side of the screen to the other.

Meanwhile, an evil machine is slowly filling the screen with yellow balloons. The player must try to hit the machine, a moving block, as many times as possible, but while breaking as few balloons as he or she can. You see, the final score is calculated by multiplying the number of hits times the number of balloons remaining.

Obviously, if one misses the square, it is good strategy to try and hit as few balloons as one can. This can be done by steering the arrow into vacant spaces left behind by balloons that have already popped. By careful timing, it is possible to hit the square

more than once on each trip across. In fact, there is one maneuver, which I will not divulge, that allows hitting the square four, five, six, or more times on one traverse. High score to date and number of hits are displayed at the end.

The game also includes a clever opening demonstration and other features that show just how sophisticated a game can be and still use only BASIC programming techniques.

In operation, the game is much like those that have gone before. The arrow character is POKEd into location N1, in routines beginning at lines 390, with regular checks to see if the balloon character (number 81 from the VIC 20 POKE table) occupies the same place. If so, control branches to the POP noise routine at 1350. A space is POKEd in the location vacated by the arrow, which will leave a hole if that space was previously occupied by a balloon.

When H does not equal a space, but does hold something other than a balloon, the player has struck the enemy box. In this case, control goes to a screen flashing routine at line 500, and the number of HITS is incremented by one.

The total points amassed are calculated beginning at 630. A FOR . . . NEXT loop from the beginning of character memory (CHAR) to the position of the player's base (PO) is commenced, and a PEEK to each location done. If that memory position holds an 81, then the number of balloons remaining (BALL) is increased by one. High scores are announced, and the player is allowed to begin once more.

```
10 REM *********
20 REM *         *
30 REM * POP ! *
40 REM *         *
50 REM *********
60 POKE36879,104
70 PRINT"(CLR)"
90 BLACK=0:WHITE=1:RED=2:CYAN=3:PURPLE=4
   :GREEN=5:BLUE=6:YELLOW=7
100 RSPACE=160:ARROW=30:SPACE=32
110 CSCREEN=37888+4*(PEEK(36866)AND128):B1=CSCREEN
120 CHAR=4*(PEEK(36866)AND128)+64*
    (PEEK(36869)AND120):B=CHAR:E=CHAR+484
130 DF=CSCREEN-CHAR
140 DD=37154:PA=37137:PB=37152
150 POKE37139,0
160 GOTO 810
```

PROGRAM LISTING 18. POP !

```
170 REM *** CHECK JOYSTICKS ***

180 POKEDD,127:S3=-((PEEK(PB)ANDl28)=0)
    :POKEDD,255
190 P=PEEK(PA):S2=((PANDl6)=0):FR=-((PAND32)=0)
200 RETURN

205 REM *** ALTER POSITION ***

220 PRINT"(CLR)":SH=CSCREEN:PO=E:HITS=0:BALLS=0
230 GOSUBl80
240 IF SH-DF>PO GOTO 630
250 IF S3=0 AND S2=0 THEN GOTO 320
260 IF F2=l THEN Nl=Nl+S2+S3:GOTO320
270 IF S3=lTHENPO=PO+l:IF PO>E+22THENPO=E+22
280 IF S2=-lTHENPO=PO-l:IF PO<ETHENPO=E
290 IF S2=-l GOTO 310
300 POKE PO,65:POKEPO-l,32:GOTO320
310 POKE PO,65:POKEPO+l,32
320 SH=SH+l
330 IFF2=0 AND FR=lTHENFL=l:LE=50:GOSUBl350
340 IF FLAG=lTHENGOSUB390
350 IF F2=lTHENGOSUB410
360 POKE SH,CYAN:POKESH-DF,RSPACE:POKE
SH-l,YELLOW:POKESH-DF-l,81
370 GOTO 230

375 REM *** MOVE ARROW ***

390 Nl=PO-22:F2=l:FL=0
400 POKE Nl,ARROW:Nl=Nl-22:RETURN
410 H=PEEK(Nl):IFH=8lTHEN GOSUBl350:GOTO430
420 IF H<>SPACETHENGOSUB490
430 POKE Nl,ARROW:POKENl+22,SPACE:POKENll+
    DF,WHITE
440 IF Nl<CHAR+22 THEN F2=0
450 Nl=Nl-22
460 POKE Nl+22,SPACE
470 RETURN

475 REM *** CHECK FOR HIT ***

490 H=PEEK(Nl):IFH=81 THEN RETURN
500 FOR N2=l TO 50
510 GOSUBl350
520 POKE36879,25
530 POKE36879,104
540 NEXT N2
550 POKE36879,104
560 F2=0
570 HITS=HITS+l
580 RETURN

590 REM *** POKE SCREEN ***
```

PROGRAM LISTING 18—CONT.POP !

```
600 POKE 7910,81
610 POKE 38630,4

615 REM *** COUNT POINTS ***

630 FOR N=CHAR  TO PO:J=PEEK(N)
640 IF J=81THENBALL=BALL+1
650 NEXT N
660 PRINT"(CLR)"
670 PRINT"(YEL)(22 SHF-Q)"
680 PRINT"       (WHI)GAME OVER"
690 PRINT"(YEL)(22 SHF-Q)(03 DWN)(BLK)"
700 PRINT" HITS:      ";HITS
710 PRINT"(DWN) BALLOONS":PRINT" REMAINING:";BALLS
720 SCR=HITS*BALLS
730 PRINT"(DWN) SCORE :";SCR
740 IF SCR>HSTHENHI=SCR:PRINT"(GRN) (DWN)
    NEW HIGH SCORE!(BLK)":HS=SCR
750 PRINT"(DWN) HIGH:";HS
760 PRINT"(DWN)       (RVS)(YEL)PLAY AGAIN?"
770 GET A$:IF A$=""GOTO770
780 IF A$="N"THEN END
790 PRINT"(CLR)":GOTO220

795 REM *** INTRODUCTION ***

810 PRINT"(CLR)"
820 PRINT"  (DWN)(RVS)USE ATARI JOYSTICK(DWN)"
830 PRINT"(YEL)(22 SHF-Q);
840 PRINT"(YEL)(22 SHF-Q);
850 PRINT"(YEL)(02 SHF-Q)OOOOO(02 SHF-Q)OOOO
    (02 SHF-Q)OOOOO(02 SHF-Q)";
860 PRINT"(YEL)(02 SHF-Q)OO(02 SHF-Q)O
    (02 SHF-Q)O(02 SHF-Q)O(02 SHF-Q)OO
    (02 SHF-Q)O(02 SHF-Q)";
870 PRINT"(YEL)(02 SHF-Q)OOOOO(02 SHF-Q)O
    (02 SHF-Q)O(02 SHF-Q)OOOOO(02 SHF-Q)";
880 PRINT"(YEL)(02 SHF-Q)OO(05 SHF-Q)O(02
    SHF-Q)O(02 SHF-Q)OO(05 SHF-Q)";
890 PRINT"(YEL)(02 SHF-Q)OO(05 SHF-Q)O(02
    SHF-Q)O(02 SHF-Q)OO(05 SHF-Q)";
900 PRINT"(YEL)(02 SHF-Q)OO(05 SHF-Q)O(02
    SHF-Q)O(02 SHF-Q)OO(05 SHF-Q)";
910 PRINT"(YEL)(02 SHF-Q)OO(05 SHF-Q)O(02
    SHF-Q)O(02 SHF-Q)OO(05 SHF-Q)";
920 PRINT"(YEL)(02 SHF-Q)OO(05 SHF-Q)O(02
    SHF-Q)O(02 SHF-Q)OO(05 SHF-Q)";
930 PRINT"(YEL)(02 SHF-Q)OO(05 SHF-Q)OOOO
    (02 SHF-Q)OO(05 SHF-Q)";
940 PRINT"(YEL)(22 SHF-Q)";
950 PRINT"(YEL)(22 SHF-Q)";
960 PRINT"(YEL)(22 SHF-Q)";
970 PRINT"(YEL)(22 SHF-Q)";
980 GOSUB500
990 PRINT"    (RVS)(DWN)COPYRIGHT 1983"
```

PROGRAM LISTING 18—CONT. POP !

```
1000 PRINTTAB(4)"(RVS)DAVID D. BUSCH"
1010 FORN=1TO1000:NEXTN

1015 REM *** POP RANDOMLY ***

1020 FOR N=1 TO 50
1030 A=INT(RND(1)*374):POKECSCREEN+A+66,32
     :GOSUB1350:FORN6=1TO50:NEXTN6:NEXTN
1040 PRINT"(CLR)(02 DWN)"

1045 REM *** INSTRUCTIONS ***

1050 PRINTTAB(4)"INSTRUCTIONS?"
1060 GET A$:IF A$=""GOTO1060
1070 IF A$="N"GOTO220
1080 IF A$<>"Y"GOTO1060
1090 PRINT"(CLR)(02 DWN)"
1100 PRINT" AN AUTOMATIC MACHINE,"
1110 PRINT" GONE AMOK, IS FILLING"
1120 PRINT" A ROOM FULL OF YELLOW"
1130 PRINT" BALLOONS. YOUR JOB IS"
1140 PRINT" TO HIT THE MACHINE AS"
1150 PRINT" MANY TIMES AS YOU CAN"
1160 PRINT" WITH YOUR ARROWS WITH"
1170 PRINT" OUT BREAKING TOO MANY"
1180 PRINT" BALLOONS."
1190 PRINT"(DWN)      (RVS)HIT ANY KEY"
1200 GET A$:IF A$=""GOTO1200
1210 PRINT"(CLR)(DWN)"
1220 PRINT" AS THE MACHINE GETS(DWN)"
1230 PRINT" LOWER, IT IS EASIER (DWN)"
1240 PRINT" TO HIT, BUT YOU MAY (DWN)"
1250 PRINT" BREAK MORE BALLOONS.(DWN)"
1260 PRINT" IF YOU MISS, STEER  (DWN)"
1270 PRINT" TO AVOID BALLOONS(DWN)"
1280 PRINT" YOUR SCORE IS TOTAL(DWN)"
1290 PRINT" HITS TIMES BALLOONS (DWN)"
1300 PRINT" REMAINING AT THE END."
1310 PRINT"(DWN)       (RVS)HIT ANY KEY"
1320 GETA$:IFA$=""GOTO1320
1330 GOTO 220

1335 REM *** MAKE NOISE ***

1350 POKE36878,15:POKE36876,255:FORG1=1
     TOLE:NEXTG1:POKE36878,0
1360 LE=0:RETURN

#
```

PROGRAM LISTING 18—CONT. POP !

CHAPTER 19

AUTO WRITER

Type: Nonsense
Size: 1000 bytes, but needs 5000 to run. For VIC 20 with expansion cartridge

Relax. The hardest part is over. The final two programs in this book are just for fun. Neither will teach you very much, other than how to find things for your computer to do when it isn't otherwise occupied. AUTO WRITER is a program designed to prove that anything monkeys can do, a computer can do faster, and nearly as well. We have all heard the saying that a gang of simians pounding at random on typewriter keyboards would eventually produce all the greatest classics ever written.

While apes are suitably random, they are a bit slow. Your VIC 20 can type well in excess of several hundred words a minute. At that rate, we can afford to have most of them turn out to be nonsense. However, true randomness would make the time needed to turn out a legitimate classic or even a run-of-the-mill best seller prohibitively long.

Instead of just choosing letters aimlessly, couldn't we improve the odds just a little, without leaving the monkey philosophy behind totally? AUTO WRITER generates random characters and punctuation, roughly in the same proportions as they appear in the English language. For example, in any given 1000 characters of text, the letter "e" appears approximately 100 times. The letter "t" will be found 77 times, and so forth.

For this program, we will consider a space to be a character (the most frequent one found, in fact), and digraphs, such as "th", "ee", and "he" to be characters as well. Periods also appear in text in predictable frequencies, so we will lump them in the same pot. If we take these characters out of the stew at random, we will get a block of text that is not truly random. It will, rather, represent the approximate mix of those characters in English.

It accomplishes this magic with a truly monstrous array, DIMensioned in line 100 to 1061 elements. We dump the alphabet into that array roughly in the same proportions that those letters appear in English, add digraphs and punctuation, and arrive at 1061 total. First, a small string array, P$(n), is filled with periods and question marks. These will be used whenever the sentence we are generating becomes longer than 20 characters. Then, starting in line 180, the frequency data is read into another array, FR(n). After the letters and digraphs are deposited in LE$(n), we start filling up the big array. A FOR . . . NEXT loop from N5 to the number equal to the frequency of that letter (this is stored in FR(n)) puts that character in A$(n).

For example, if, according to the data in the frequency array, a letter should appear 37 times, the loop will repeat 37 times, and N5 will count off that many to ensure that the next character will be placed in the next element of the array A$(n). This all takes place within two nested loops at line 240–290. Then, actual generation of text begins. The variable R is repeatedly given a new random value (line 310), ranging from 0 to 1060, and the R element of A$(n) added onto T$, which is the text string.

If T$ is shorter than 20 characters, the program simply prints the new character to the screen, and goes back for more random characters. If T$ is longer, a check is made to see if the last character entered is a space (CHR$(32)). If so, a new variable G is selected randomly, in the range 1–6, in line 380. The G element of the punctuation array, (P$(G)), is printed to the screen, along with two spaces. This ends the sentence. T$ is nulled, and the program loops back to line 310 for more random character selection.

Although the process takes a few minutes to describe, rest assured that your VIC 20 will be able to print text to the screen as fast as you can read. The problem, of course, is recognizing a major novel when one appears. You could sit at the screen for

hours watching and waiting. Or, you could add two program lines:

```
335 IF T$ = "Hamlet: Prince of Denmark, Act The First"
    GOTO 530
530 PRINT" THIS IS IT!!!!!!!"
```

Obviously, attaching a printer could go through a lot of paper very fast. I'd recommend just running the program for a few minutes at a time, and seeing what turns up. It may not be great literature, but it is fun.

```
10 REM ***************
20 REM *             *
30 REM * AUTO WRITER *
40 REM *             *
50 REM ***************

55 REM *** MAKE LAME EXCUSE ***

60 PRINT"(CLR)(02 DWN"
70 PRINT"    (RVS)AUTO WRITER"
80 PRINT"(02 DWN)PLEASE BE PATIENT"
90 PRINT"(DWN)   I AM THINKING."
100 DIM A$(1061),FR(40),LE$(40)
110 B=1

115 REM *** READ CHARACTER FREQUENCY ***

120 FOR N1=1 TO 4
130 P$(N1)=CHR$(46)
140 NEXT N1
150 FOR N2=5 TO 6
160 P$(N2)=CHR$(63)
170 NEXT N2
180 FOR N3=1 TO 37
190 READ FR(N3)
200 NEXT N3
210 FOR N4=1 TO 37
220 READ LE$(N4)
230 NEXT N4
240 FOR J=1 TO 37
250 FOR N5=B TO FR(J)+B-1
260 A$(N5)=LE$(J)
270 NEXT N5
280 B=N5
290 NEXT J

295 REM *** BEGIN PRINTING ***

300 PRINT"(CLR)(02)DWN"
310 R=INT(RND(1)*1060)
```

PROGRAM LISTING 19. AUTO WRITER

```
320 T$=T$+A$(R)
330 T=LEN(T$)
340 T$=LEFT$(T$,LEN(T$)-1)
350 IF T<20 GOTO 430

355 REM *** END SENTENCE IF SPACE ***

360 IF A$(R)<>CHR$(32) GOTO 430
370 IF RIGHT$(T$,1)<>CHR$(32) GOTO 430
380 G=INT(RND(1)*6)+1
390 T$=""
400 PRINT P$(G);
410 PRINT "  ";
420 GOTO 310
430 PRINT A$(R);
440 GOTO 310

445 REM *** FREQUENCY DATA ***

450 DATA 310,50,31,47,54,20,46,38,30
460 DATA 4,18,20,23,23,23,15,15,15
470 DATA 15,15,38,31,18,17,17,16,14
480 DATA 14,13,12,12,12,8,7,6,5,4
490 DATA " ",E,T,A,O,N,I,R,S,H
500 DATA D,L,F,C,M,U,G,Y,P,B
510 DATA TH,HE,ER,AN,IN,ON,RE,AT
520 DATA ED,ST,ND,ES,LL,EE,SS,OO,TT

#
```

PROGRAM LISTING 19—CONT. AUTO WRITER

CHAPTER 20

URBAN RENEWER

Type: Nonsense
Size: 2400 bytes, for any size VIC 20

URBAN RENEWER is another program that you never knew you needed. After you run it, you may still feel the same way. It is another just-for-fun program that will randomly produce the names of a few dozen likely sounding cities, metropolises, and hamlets quicker than you can say "Great Gotham, Batman!"

As a special added, no cost attraction, the program will also produce names for those Spanish-sounding towns in Florida or California that you would never visit if you knew what they meant in English. After all, although it SOUNDS beautiful, Boca Raton really means "rat's mouth."

Choosing city names at random is a bit trickier than generating Pulitzer Prize quality fiction. MXYZPTVILLE is highly unacceptable as a city name outside Wales. So, while we will choose suffixes and prefixes at random, they will be selected from a handy list provided in DATA lines within URBAN RENEWER itself. Best of all, you can add your own to enhance the intrinsic worth of this program up past null to nil. Instead of having the FOR . . . NEXT loops read from 1 to a constant, this program defines the upper limits of the various loops as variables.

Three variables, SUF, PREF, and SP are used to stand for the number of suffixes, prefixes, and Spanish words contained in the DATA lines. If you add any of your own, simply append them to the end of that type of word fragment in the DATA, and then

change the appropriate variable in line 60. Everything else has been taken care of. For example, the DIM statement in line 70 uses the three variables to set up the string arrays that store the data. As mentioned, the FOR . . . NEXT loops are also adjusted automatically.

At the risk of providing a bit more education this late in the book, use of variables in this manner can greatly simplify modifying a program later. One change of a variable's definition fixes all uses of that variable in the program.

The VIC 20 also operates somewhat faster with variables instead of constants. It checks an internal variable list, which is sorted in the order in which the variables are used in the program. Since SUF, PREF, and SP are defined right at the start, it finds them rather quickly. When using constants, on the other hand, it must work a bit more slowly, particularly if the constants have not been defined as integers. As you have probably surmised, the city names are put together by combining a prefix with a suffix, chosen at random.

If you type in the DATA lines, be careful to include spaces BEFORE all the names that are shown within quotes. These will ensure that a city like "PARK WOODS" will be displayed as two words, and not as the less desirable "PARKWOODS". Such data must be included inside quotes, or else the VIC 20 will ignore the space.

The modulo arithmetic trick is used in line 680 to ensure that just 15 names will be displayed on the screen at once. The only difference between the routines that produce English city names and those that produce the Spanish names is that the Spanish routine does not distinguish between prefixes and suffixes. To the confused tourist, LINDA PLAYA and PLAYA LINDA sound equally likely. I did take the clever step of comparing, in line 760, the number chosen for the prefix with the one chosen for the suffix, to make sure they are not the same. Even the least astute visitor would be suspicious of PLAYA PLAYA. That is, unless he or she lives in or came from Walla Walla, Washington.

```
10 REM *****************
20 REM *                 *
30 REM * URBAN RENEWER *
40 REM *                 *
50 REM *****************
```

PROGRAM LISTING 20. URBAN RENEWER

```
60 SUF=58:PREF=48:SP=24
70 DIM SUF$(SUF),PREF$(PREF),SPAN$(SP)

75 REM *** READ SUFFIXES ***

80 FOR N=1 TO SUF
90 READ SUF$(N)
100 NEXT N

105 REM *** READ PREFIXES ***

110 FOR N1=1 TO PREF
120 READ PREF$(N1)
130 NEXT N1

135 REM *** READ SPANISH NAMES ***

140 FOR N2=1 TO SP
150 READ SPAN$(N2)
160 NEXT N2

165 REM *** DATA ***

170 DATA CHESTER," RIVER"," WOODS"
180 DATA " MILLS",CREST," PARK",WORTH
190 DATA WICK," GROVE",GLEN,MORE
200 DATA HAM,ROSE,LAWN,MOUNT,HURST
210 DATA TON,WOOD,RIDGE,FORD,LAND
220 DATA PORT,SIDE," HILL",DALE
230 DATA WATER,FIELD,HOPE,SPRINGS
240 DATA POND,SHORES,POINT,MONT
250 DATA " HEIGHTS",STONE,CREEK,STEAD
260 DATA DON,BRIER," BEACH",FELD
270 DATA ELLE,IANA,MAN,BURG,BORO
280 DATA OPOLIS,TOWN," CENTER"," FALLS"
290 DATA VILLE," VILLAGE"," CITY"
300 DATA AVIA,OVIA,INGTON," CORNERS"
310 DATA FAIR,GARDEN,KING,GRAND,TON
320 DATA HOME,JAMES,LEEDS,LIME
330 DATA MID,NEW,OAK,COTTON,LAKE
340 DATA BRICK,HIGH,CEDAR,SANDY
350 DATA EDGE,BRIDGE,CIRCLE,ROSE
360 DATA MARKET,SUGAR,SHADY,MAPLE
370 DATA BELLEVUE,POPLAR,CLARK
380 DATA SPRING,FOREST,SLEEPY,PLAIN
390 DATA WATER,RICH,CENTRAL,EAST
400 DATA ASH,GLEN,ROCK,LINCOLN,BAY
410 DATA YORK,MEADOW,NORTH,SOUTH
420 DATA WEST,SANDY,NOR
430 DATA HERMOSA,COSTA,DORADA,MESA
440 DATA BUENA,SERRANA,PUERTA,ISLA
450 DATA LAGUNA,PUEBLA,ALTA,RIO
460 DATA SANTA,VIEJA,PASA,PICA
470 DATA ROSA,CARA,BONITA,BOLSA
480 DATA PLAYA,LINDA,MIRADA,BOCA
```

PROGRAM LISTING 20—CONT. URBAN RENEWER

```
490 PRINT"(CLR)     (RVS)URBAN RENEWER"
500 PRINT"(02 DWN)  ENTER CHOICE:"
510 PRINT"(02 DWN) (RVS)1.(OFF) ENGLISH NAMES"
520 PRINT"(02 DWN) (RVS)2.(OFF) SPANISH NAMES"
530 GET A$:IF A$=""GOTO 530
540 A=VAL(A$)
550 IF A<1 OR A>2 GOTO 530
560 PRINT"(CLR)(02 DWN)"
570 PRINTTAB(2)"HOW MANY NAMES"
580 PRINTTAB(2)"DO YOU WANT"
590 INPUT AN$
600 PRINT"(CLR)(02 DWN)"
610 AN=VAL(AN$)
620 ON A GOTO 630,720

625 REM *** PRINT ENGLISH NAMES ***

630 FOR N=1 TO AN
640 S=INT(RND(1)*SUF)+1
650 PRINT"  ";
660 P=INT(RND(1)*PREF)+1
670 PRINT PREF$(P);SUF$(S)
680 IF N/15=INT(N/15)THEN GOSUB 820
690 NEXT N
700 GOSUB 820
710 GOTO 490

715 REM *** PRINT SPANISH NAMES ***

720 FOR N=1 TO AN
730 S=INT(RND(1)*SPAN)+1
740 PRINT"  ";
750 P=INT(RND(1)*SPAN)+1
760 IF P=S GOTO 750
770 PRINT SPAN$(P);" ";SPAN$(S)
780 IF N/15=INT(N/15)THEN GOSUB 820
790 NEXT N
800 GOSUB 820
810 GOTO 490
820 PRINT"(DWN) HIT ANY KEY"
830 PRINT" TO CONTINUE"
840 GET A$:IF A$=""GOTO 840
850 PRINT"(CLR)(02 DWN)"
860 RETURN

#
```

PROGRAM LISTING 20—CONT. URBAN RENEWER

How To Load These Cassette Programs Into Your VIC 20

If at some future time you decide to purchase the prerecorded tape of the programs in the book, the tape should be loaded according to the following instructions.

This tape will load according to the LOAD instructions in your VIC 20 user's manual (the one that came with your computer). If you don't have the manual, here's how to do it:

1. Connect and power up your VIC 20, a TV screen, and the VIC 20 Datasette. Open the door of your Datasette and insert the program cassette in the same way you would an audio cassette. Close the door. Press no buttons yet.
2. Suppose the name of the program you want to load is NIFTY. Using your VIC 20 keyboard, type **LOAD "NIFTY"** just as you see it here, including the space and quotes. Press RETURN.
3. Your VIC will respond on-screen with **PRESS PLAY ON TAPE.** You then depress the "Play" button on the Datasette.
4. The tape starts, your VIC displays **SEARCHING FOR NIFTY,** and the tape continues to roll. As each program passes through, VIC displays **FOUND** followed by the program name. When NIFTY arrives, VIC displays **FOUND NIFTY** and then shows the single word **LOADING.** When the tape stops, the program will have been loaded into RAM and the screen will say **OK READY.** You're ready to RUN and enjoy your program at any time.
5. When you're ready to load another program, repeat the process, using the name of the new program. Your VIC will automatically delete the old program from memory as it loads the new one. The program on the tape, of course, remains undisturbed and can be reused many times.

Alternate Ways To Load From Cassette

If the program you want to load is one of the last ones on the tape, you can get pretty bored waiting for the others to pass through before yours comes up. To avoid this, set the tape counter on your Datasette to zero and then start the loading process. Note the counter reading as each program starts through and write the number in your book. If you make it a habit to zero the counter each time you insert a cassette, you can use your Fast Forward button to quickly run the tape ahead until

the counter shows the index number for the start of the program you want. Then LOAD it by name as described earlier.

A similar — but less accurate — way is to use the index marks molded into the cassette body just under the window between the spindle holes. By noting the index mark closest to the rim of the tape, you can get a rough idea of where each program starts and go from there.

If you don't care about time, just insert the cassette, type LOAD with no program name following, and press RETURN. Your VIC will load the first program on the tape, and then load each succeeding program each time you retype the command. This is a good way to explore the tape as well as to get the tape-counter index number as each program starts.

Programs on This Tape, With Locations

Computer ESP	000	Bingo	192
Mind Reading	015	Floor Planner	202
Auto Cost	030	Cookie Shop	210
Space Command	050	Black Book	227
Album Timer	094	Motor Race	236
Cost Estimator	106	VIC Organ	251
Reaction Timer	127	Barrier Run	263
Stock Market	142	POP !	274
Bulletin Board	163	Auto Writer	292
Kitchen Timer	179	Urban Renewer	300

More SAMS VIC 20 Books You Can Enjoy

VIC-20: 50 EASY-TO-RUN COMPUTER GAMES

A collection of games for the unexpanded VIC 20, most of which have 15 to 20 program statements. By Edward Burns.

BOOK ONLY: 96 pages, 5½ x 8½, soft. ©1983.
Ask for No. 22188 . . . . . $5.95

TAPE CASSETTE OF PROGRAMS ONLY
Ask for No. 26510 . . . . . $7.95

BOOK PLUS TAPE CASSETTE: Packed in 6 x 9 hardcover vinyl binder with cassette storage feature.
Ask for No. 26170 . . . . . $12.95

VIC 20 PROGRAMMER'S NOTEBOOK

A collection of VIC 20 programming techniques, hints, kinks, tricks, subroutines, and shortcuts. By Earl R. Savage. 256 pages, 5½ x 8½, comb. ©1983.

Ask for No. 22089 ..... $14.95

VIC 20 PROGRAMMER'S REFERENCE GUIDE

Covers VIC 20 BASIC, machine language, I/O operations, programming tips, advice, and more. By Commodore Computer. 289 pages, 5½ x 8½, comb. ©1982.

Ask for No. 21948 ..... $16.95

These and other Sams Books and Software products are available from better book and software retailers worldwide, or directly from Sams. To order, call 800-428-SAMS or 317-298-5566 and charge your selection to your MasterCard or VISA account.

HATE TO KEY-IN PROGRAM LISTINGS?

Spend $7.95 for a ready-to-run tape of the twenty programs in *VIC-20 GAMES, GRAPHICS, AND APPLICATIONS* and you'll avoid spending hours tracking down a bug accidentally typed into a stalled or crashed program (it can happen to *anybody*)!

Ask for No. 26513 ..... $7.95

Special combination package — book and tape together!

Ask for No. 26167 ..... $15.95

Use the handy order form and send to the address below or call the Sams Order Desk at 800-428-3696 or 317-298-5566 and charge it to your MasterCard or Visa account.

Ask about other Sams book/tape combinations for the VIC-20, Commodore 64, and other microcomputers too, or contact your local Sams dealer.

Howard W. Sams & Co., Inc.
4300 WEST 62ND ST. INDIANAPOLIS, INDIANA 46268 USA

Catalog No.	Qty.	Price	Total	Catalog No.	Qty.	Price	Total

Subtotal	
Add local tax where applicable	
Add Handling Charge	2.00
Total Amount Enclosed	

☐ Check ☐ Money Order ☐ Visa
☐ MasterCard

Account Number _____ Expires _____

Name (print) _____

Signature _____

Address _____

City _____ State _____ Zip _____

Prices subject to change without notice. Offer good in USA only. In Canada, contact Lenbrook Electronics, Markham, Ontario L3R 1H2.

X0459

SAMS

VIC 20™

Games, Graphics, and Applications

- The Commodore VIC 20 has many special features, including:
  - — user-definable character sets
  - — four musical voices
  - — a real time clock
  - — color
  - — graphics
- This book teaches you how to use all of these major features through 20 programs designed for the 5K, and expanded VIC 20.
- All features and BASIC statements are thoroughly explained.
- It's great for new VIC 20 users . . . but advanced VIC programmers gain much benefit, too!
- The book also contains joystick games and programs for home applications.

Howard W. Sams & Co., Inc.
4300 West 62nd Street, Indianapolis, Indiana 46268 U.S.A.

$8.95/22189 ISBN: 0-672-22189-6

SAMS

VIC 20

Games, Graphics, and Applications

- The Commodore VIC 20 has many special features, including:
  - — user-definable character sets
  - — four musical voices
  - — a real time clock
  - — color
  - — graphics
- This book teaches you how to use all of these major features through 20 programs designed for the 5K and unexpanded VIC 20.
- All features and BASIC statements are thoroughly explained.
- It's great for new VIC 20 users . . . but advanced VIC programmers gain much benefit, too!
- The book also contains joystick games and programs for home applications.

Howard W. Sams & Co., Inc.
A Publishing Subsidiary of ITT
4300 West 62nd Street, Indianapolis, Indiana 46268 U.S.A.

$15.95/26167

ISBN O-672-26167-7